Final Master Thesis

MASTER'S DEGREE IN AUTOMATIC CONTROL AND ROBOTICS

Autonomous Vehicle navigation with Deep Reinforcement Learning

MEMORY

Author: Àlex Cabañeros

Advisor: Prof. Dr. Cecilio Angulo

Date: April, 2019

Escola Tècnica Superior

Enginyeria Industrial de Barcelona

Abstract

The irruption of Autonomous Vehicles in transportation sector is unstoppable. However, the transition from conventional vehicles to Autonomous Vehicles will not happen from one day to the other, instead, it will be a process of several years in which, gradually, new autonomous/automated functionalities will be equipped to the vehicles and introduced to the customers. These automated/autonomous functionalities are, nowadays, known as ADAS (Advanced Driver Assistance Systems).

The aim of this project is, through the combination of different ADAS functions, make the vehicle navigate on a highway autonomously, but at the same time, following the traffic rules and regulations requirements, and also guaranteeing the safety on the road. In order to accomplish this objective, the proposed approach is to implement the Policy Gradient Reinforcement Learning method to select the proper function at each moment.

The actual regulatory framework of the road safety will be explained in order to understand the ADAS functions that the model will combine, as well as to know how these will evolve in the near future.

The algorithm will be tested using a five-lane highway simulator, previously selected after a study of the state-of-the-art of Autonomous Vehicles' simulators. The vehicle will be guided by LIDAR data coming from the sensor installed in the vehicle (sensor equipped in most of the future Autonomous Vehicles).

Results and performance of the model through experimentation will be presented and evaluated using the simulator, as well as the different network morphologies.

CONTENTS

ACRONYMS

ACSF Automatically Commmanded Steering Function.

ADAS Advanced Driver Assistance Systems.

AI Artificial Intelligence.

AV Autonomous Vehicle.

CEL Complex Electronic Systems.

CSF Corrective Steering Function.

EC European Commission.

ELU Exponential Linear Unit.

ESF Emergency Steering Function.

GR Working Parties.

GRVA Working Party on Automated/Autonomous and Connected Vehicles.

LC Lane Change.

LIDAR Light Detection and Ranging.

ML Machine Learning.

PA Parking Assist.

PG Policy Gradient.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

SAE Society of Automotive Engineers.

TF TensorFlow.

UNECE United Nations Economic Commission for Europe.

WP29 World Forum for Harmonization of Vehicle Regulations.

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Mobility is one of the most challenging topics that society will face up to the coming years. It will not only entail a technological challenge, but also a social change due to the paradigm shift of transport modes and its improvements on public issues.

Nowadays, society is confronting health problems mostly caused by transportation; traffic deaths have become the 10^{th} cause of death in the world, and the first cause for young people [1]. Besides, the injuries caused by traffic accidents have increased the recent years. Clearly, other problems derived from transportation are in terms of Global Warming caused by the gas emissions of these. In some countries it can reach up to the 28% of the total emissions that cause the Greenhouse effect [2]. Moreover, another important problem originated by the vehicles is the time spent by drivers and passengers in traffic jams or looking for parking lots. In the case of Barcelona, people are spending 148 hours a year in congestion [3].

Most of these above problems are a consequence of the rapid increase of particular vehicles and, therefore, the also growth of total drivers. It is also a significant datum that the 94% of traffic death are attributed to the driver [4] and not to the vehicle or the infrastructure.

For this reason, manufactures have positioned their future's spotlight in Autonomous Vehicle (AV) development. AV will reduce the number of traffic accidents (optimal decision will be taken, communication with other vehicles, etc.), reduce traffic jams and hours wasted inside the car, will optimize the energy consumption reducing gas emissions, etc. Consequently, it will not only be a technological challenge where companies will have to invest in new technologies and development, but a social challenge, because society will have to change the way they interact and spend time on these vehicles.

Even though autonomous vehicles will play an important role in the future change of the mobility, which will have a considerable impact on society (although benefits and disadvantages are still hypothetical), manufacturers will not suddenly take the product in to the market. Doubtlessly, it will be introduced step by step, because it not only needs to be accepted by society, but also by the regulatory bodies that ensure safety on the roads.

The first part of this project will consist on the study of the current regulatory framework (regulations, regulatory bodies, countries, organizations, manufacturers, etc.), in order to know how the manufacturers are facing the development of their autonomous vehicles.

Nowadays, there already exist different autonomous/automated functions that allow the vehicle to perform certain actions by itself while the driver is only monitoring it. Depending on how these functions control the vehicle, there has been defined six levels of driving automation [5] (from no automation to fully automation) Figure 1.1 by the SAE:

- SAE level 0: Zero autonomy; the driver performs all the driving tasks.
- SAE level 1: Vehicle is controlled by the driver, but some driving assists features may be included in the vehicle design.
- SAE level 2: Vehicle has combined automated functions, like acceleration and steering, but the driver must remain engaged with the driving task and monitor the environment at all times.
- SAE level 3: Driver is a necessity, but is not required to monitor the environment. The driver must be ready to take control of the vehicle at all times with notice.
- SAE level 4: The vehicle is capable of performing all driving functions under certain conditions. The driver may have the option to control the vehicle.
- SAE level 5: The vehicle is capable of performing all driving functions under all conditions. The driver may have the option to control the vehicle.

As it has been stated before, the automated functions are known as Advanced Driver Assistance Systems (ADAS) . These are some examples already in the market ADAS systems:

- Adaptive Cruise Control
- Collision Avoidance Systems
- Lane Change Assist
- Blind Spot Detection

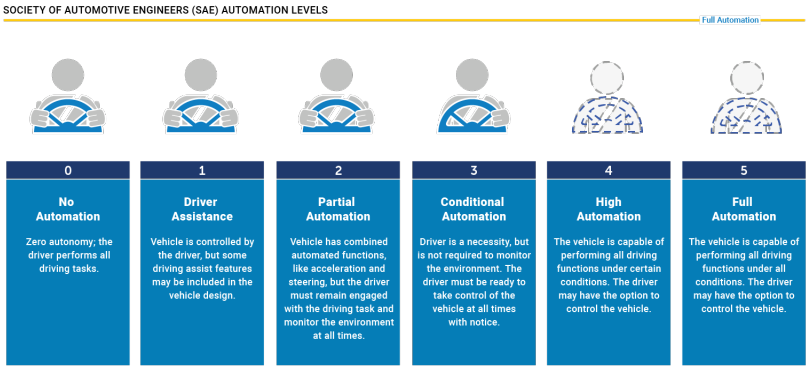

Figure 1.1: SAE levels of driving automation

- Lane Keeping
- etc.

In order to ensure safety, ADAS function has been developed under the guidelines of the regulatory bodies. This is the reason why most of the functions already commercialized have operational boundaries (they can only be used in roads that fulfils certain conditions, they have operational times, etc.) and, functions from different manufacturers, behave highly similarly one to each other.

Furthermore, one way to achieve the complete autonomous control of the vehicle, may be by the combination of the different ADAS functions that the vehicles are equipped. By doing so, the vehicle can be classified from a vehicle of SAE level 1 to SAE level 4/5.

The project was set out in a way that, in order to achieve an autonomous navigation, it would be done by doing so: the combination of existing ADAS functions. The vehicle will have the task to know how to coordinate lateral and longitudinal functions of the vehicle in order to navigate safely without human intervention. The aim of using already existing ADAS functions is to try to bring closer the development of an autonomous vehicle (SAE 4-5) with the state-of-the-art of ADAS functions and the regulatory framework.

Due to the fact that most of the ADAS functions responsible of the lateral and longitudinal movement of the vehicles can only work on highways, the second part of the project will be a research of the different simulators of autonomous vehicles, and specially, on those that simulates a highway with several lanes.

Finally, the brain of the vehicle, responsible of choosing the correct ADAS function at each state, will be based on a Reinforcement Learning (RL) method. A LIDAR sensor will be the responsible of telling the state of the vehicle at each moment. The implementation will be achieved by a research of the state-of-the-art of the RL methods, and in particular those based on Policy Gradient (PG).

1.1 MOTIVATION

Autonomous Vehicles will disruptively change the way we nowadays understand mobility. They will not only solve or reduce most of the problems stated above (traffic deaths, pollution, traffic jams, etc.) but it will also change the role of the persons inside the vehicle: from drivers and passengers to just users. These users will not longer be responsible for the dynamics of the vehicle, but they will be able to spend the time inside the vehicle freely. For this reason, a personal motivation of this project is that by contributing a quick implementation of AV would help to reduce the global problems stated above.

The development of AV is subjected to several technological challenges (Artificial Intelligence (AI), new sensors, etc.), but also to regulation challenges. Traditional manufacturers (DAIMLER, VW Group, Ford, FIAT, etc.) and new players (Waymo, UBER, Apollo, etc.) need to take in to account both aspects in the development of their systems in order to be able to launch the product to the market.

The main motivation of the project is the development of an autonomous system for a vehicle through the implementation of a RL method, but at the same time, keeping the development as close as possible to the actual regulatory requirements.

The idea of the project was born after working in Applus IDIADA Automotive SA, specifically, in the Homologation Department. While working there, I could see how vehicle's manufacturers had the obligation to have a deep knowledge on how the regulations were being developed, in order to design their futures products. Otherwise, if the requirements the regulations asked were not met, manufacturers would not be able to launch the products to the market.

With the knowledge about the homologation's framework and the state of the art of the manufacturer's ADAS development, and after having taken some Artificial Intelligence lectures during the Master, the idea was to combine them all to develop an AV.

Thus, the implementation would be a huge challenge in which it could be use all the knowledge and tools we learned during the Master and, at the same time, the knowledge obtained during the internship in IDIADA.

1.2 OBJECTIVES

The objective of the project is to explore and develop a RL algorithm in order to make a vehicle able to travel through a multiple lane road with traffic, maximizing the speed and complying with safety driving parameters. Different autonomous vehicle simulation environments will be evaluated and compared, and then one will be selected to develop the project. Because of the complexity and wide range of possibilities that the general objective presents, the project has been divided in sub-objectives. This section will present the sub-objectives that have been needed to achieve the main objective:

1. Regulatory framework
    - Study of the regulatory bodies. How they are addressing the future of AV
    - Study of the regulations related with ADAS systems. How they are now, and how they will be addressed in the future
2. Simulator
    - State-of-the-art of AV simulators, comparison and adequate selection the one that best fits with the project's goals
    - Study of the simulator selected. Understand how to interact with the simulator and how to get all the data necessary
    - Implementation of the simulator in the environment of work available (implementation Guidelines)
3. Reinforcement Learning
    - Introduction to RL
    - Explanation of PG algorithm
    - Implementation of the algorithm using Python and test it with other environments
    - Implementation of the algorithm together with the AV simulator
4. Future work: Document implementations and present results in order to facilitate future work and improvements

In parallel, practical skills will be improved in order to implement the project. These are:

- Python: the use of Python will be needed in order to implement not only the simulator, but TensorFlow (TF). The use other libraries of Python as Numpy, Matplotlib, etc. will be needed
- TensorFlow: Google created open source library will be used to implement the RL algorithm
- OpenAI gym: different GYM environments will be used to learn about TF and to test our PG algorithm before implementing it to the simulator

1.3 SCOPE

The scope of the project is to have a PG algorithm running in a simulator, making a vehicle navigate autonomously as fast as the road permits and assuring safety conditions.

The project, as it has been stated above, has a main objective, but has been split in specific objectives, depending on the phase of development, in order to achieve the general goal:

- 1st phase: Regulatory framework's section will be a study of how the Road Safety institutions are structured, and which are the working groups responsible in approving or rejecting the regulations. It will be focused on the United Nations Economic Commission for Europe (UNECE) and European Commission (EC), due to these are the organizations responsible in Europe.
  There will be also a study of the regulations, but bounded to those related to ADAS systems.
- 2nd phase: In the second part of the project, where several simulators will be evaluated and then, one selected in order to implement the algorithm in it, the simulator must fit these basic features:
  - Be able to simulate a highway with more than one lane and at the same direction of travel, and be able to simulate more vehicles on the road
  - Be able to run the simulator in our laptop
- 3rd phase: The third part will be the implementation of a PG and the implementation of it in the simulator. And finally, verify that it allows the vehicle to navigate autonomously in the highway, maximizing the velocity but, at the same time, assuring road safety.

2 REGULATORY FRAMEWORK

Automated vehicles are foreseen to be a regulatory challenge due to its complexity of testing its functional safety and its intended functionality. Nowadays, manufacturers are developing cutting-edge ADAS functionalities which are not under the scope of the regulatory framework yet, and would make the vehicle scale to a more automated SAE level. However, due to regulatory bodies do not legislate at the same velocity as the growth of the technology, the implementation of these new functionalities in the market are slowed down.
Additionally, old but especially new manufacturers, who are developing AV of SAE level 4/5 that will be ready to be released to the market in a "near" future, are pushing the regulatory bodies to tackle this as soon as possible. In consequence, UNECE, EC and other regulatory bodies are now adjusting its structures and Regulations to be adapted to the incoming technologies.

This section will first introduce how the regulatory bodies are nowadays structured, and who are the main participants. The explanation will be focused on European countries, thus UNECE and EC institutions only. Then, a study of the Regulations in-force related to AV will be done and explained. Finally, a vision towards the future of the Regulations will be given.

In summary, this section will help the proposal of the project by knowing and understanding the already approved ADAS functions, which will be used as the functions to be combined to make the vehicle autonomous, but also to understand how it is going to be tackled future AV in order to be able to travel on public roads.

2.1 HOMOLOGATION AND REGULATORY STRUCTURE

Homologation, is defined as the process of certifying or approving a product to indicate that it meets regulatory standards and specifications, such as safety and technical requirements.

The homologation in the vehicles and components' field is the administrative procedure for which an Approval Authority verifies that a vehicle or a component fulfils with the regulatory requirements before navigating on public roads. The homologation shall follow national and international Regulations (Country Ministry laws, EC Directives, UNECE Regulations, etc.).

Approval Authorities are the official organism of each country belonging to the European Union, responsible for assuring the compliance with the EC Directives. For instance, in Spain the Approval Authority is *Ministerio de Industria, Comercio y Turismo.*

There exist a framework Directive 2007/46 establishing a framework for the approval of motor vehicles and their trailers, and of systems, components and separate technical units intended for such vehicles [7]. All the Regulations need for a Type Approval of a vehicle are gathered in this Directive. Type approval describes the process applied by national authorities to certify that a model of a vehicle meets all EU safety, environmental and conformity of production requirements before authorising it to be placed on the EU market [8].

Most of the Directives are based on the Regulations made by the UNECE in the World Forum for Harmonization of Vehicle Regulations (WP29) Figure 2.1:
WP29 offers a unique framework for globally harmonized Regulations on vehicles. The benefits of such harmonized Regulations are tangible in road safety, environmental protection and trade[6]. It is at the same time, the most important group at an international level working on incorporating into the regulatory framework the technological innovations of vehicles.

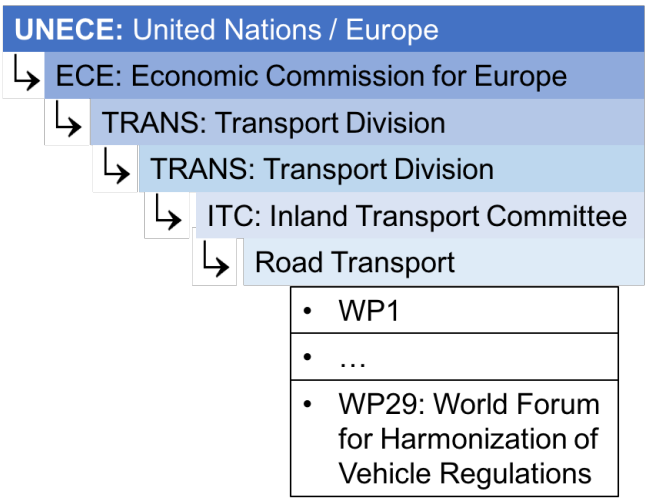

Figure 2.1: UNECE Structure

The participants of this rule-making group are any member country of the United Nations and any

regional economic integration organization, set up by country members of the United Nations. Not only countries' representatives participate in the working groups, but also manufactures, organizations, etc.

WP29 established six permanent Working Parties (GR) Figure 2.2, i.e. subsidiary bodies that consider specialized tasks, consisting of people with a specific expertise. These Working Groups are, in the areas they are specialized in, in charge to develop new Regulations and to amend the in-force ones.

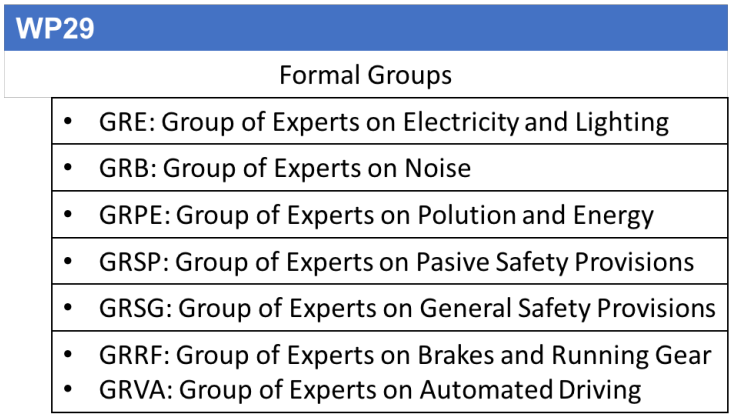

Figure 2.2: UNECE - WP29 Structure

As it can be seen, there are five GR plus a block where there are two Formal Groups: GRRF and GRVA. This is because in June 2018, foreseen the unstoppable future of AV, WP29 decided to modify GRRF group to a new group named GRVA, where only topics related to AV would be discussed. Figure 2.3 shows the Informal Groups and Task Forces inside GRVA Working Groups responsible to deal not only with new ADAS functions, but also with the future of the homologation related to AV.

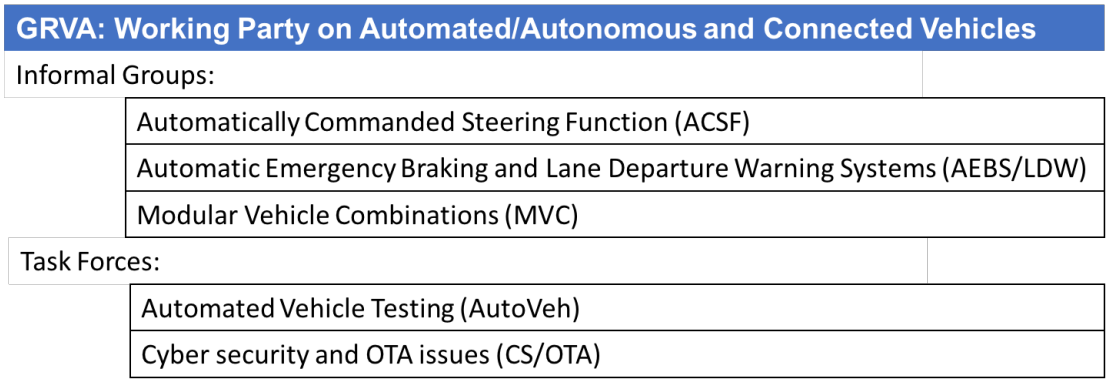

Figure 2.3: GRVA Structure

In the next section, these Informal Groups will be explained and associated, each of them, to their specific Regulation.

2.2 ADAS IN REGULATIONS

In this section, a review of the Regulations that may have an impact on the controllability of the vehicle's project will be done. In concrete, it will help to select those ADAS functions, already discussed by the regulatory bodies, that allow the vehicle to perform longitudinal and lateral movements.

As it has been said, Regulations set the requirements vehicles or components need to fulfil in order to guarantee road safety and, therefore, obtain the approval. According to UNECE, a UN Regulation *contain provisions (for vehicles, their systems, parts and equipment) related to safety and environmental aspects. They include performance-oriented test requirements, as well as administrative procedures. The latter address the type approval (of vehicle systems, parts and equipment), the conformity of production (i.e. the means to prove the ability, for manufacturers, to produce a series of products that exactly match the type approval specifications) and the mutual recognition of the type approvals granted by Contracting Parties.*

Since the creation of a common Safety Road Agreement, in 1958, all Regulations set requirements for functions or components: brake (longitudinal movement), steering (lateral movement), lights, emissions, etc. This way, ADAS functions are being incorporated in the regulatory framework by amending old Regulations that control the vehicle in the same direction, or by creating new Regulations for specific ADAS functions.

UN-R79 was historically the Regulation in charge of the steering system of the vehicle (control of the lateral movement). However, since the irruption of ADAS, and in particular those related to the lateral movement (Parking Assist, Lane Keeping, Lane Change, etc.), gradually, have been incorporated to this Regulation under to name of Automatically Commmanded Steering Function (ACSF). Figure 2.4 shows the evolution of UN-R79 (sub index after the Regulation number R79.XX is the amend number).

The first amendment, UN-R79.01, was done in order to set the ground of ADAS functions by stating and defining that in the near future will be some Advanced Driver Assistance Steering Functions that

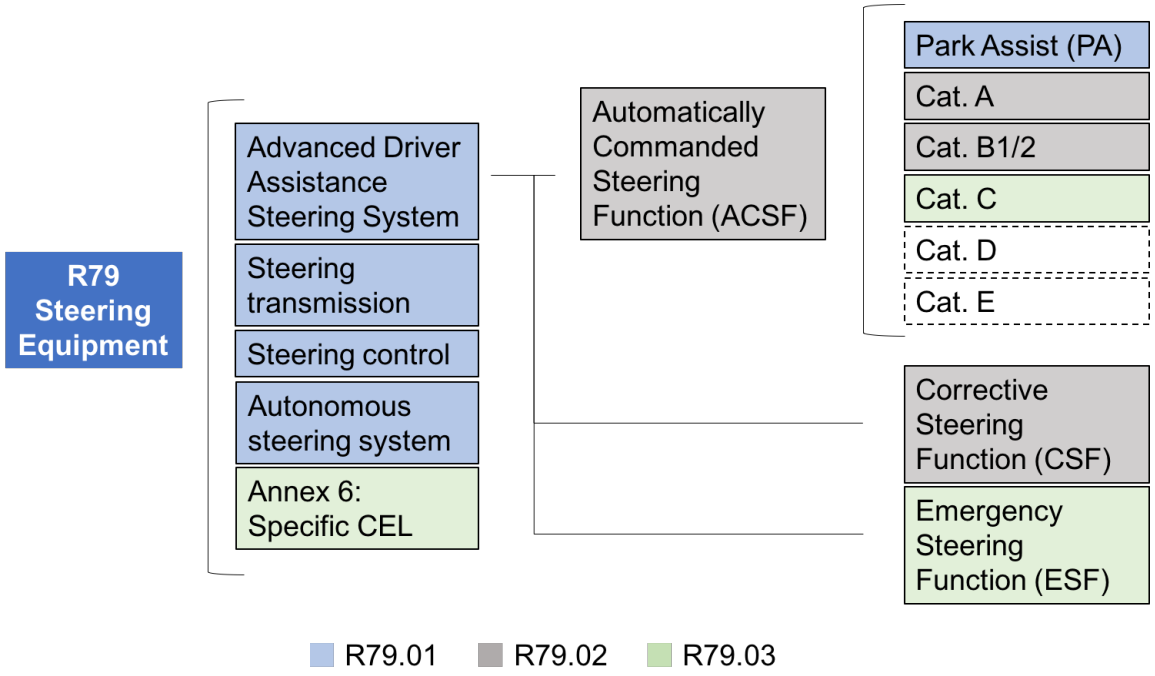

Figure 2.4: Evolution of UN R79

would be incorporated in the future. So this first amend included a description of what would mean *autonomous steering* in the field of steering the vehicle and, at the same time, add the definition and requirements for the first ADAS function that manufacturers had already developed related with steering: Parking Assist (PA).

Later, as new ADAS were being developed, a second amend was done. This time it included a new definition of ADAS in the field of steering: ACSF. Besides some other functionalities were described, as well as its requirements:

- Category A: means a function that operates at a speed no greater than 10 km/h to assist the driver, on demand, in low speed or parking manoeuvring
- Category B1: means a function which assists the driver in keeping the vehicle within the chosen lane, by influencing the lateral movement of the vehicle
- Corrective Steering Function (CSF): means a control function within an electronic control system whereby, for a limited duration, changes to the steering angle of one or more wheels may result from the automatic evaluation of signals initiated on-board the vehicle

Finally, in the third and last amend, not only new functions were added (ACSF Category C and Emer-

gency Steering Function (ESF)), but also a new and more exhaustive evaluation of the functional safety and intended functionality (Annex 6: Specific Complex Electronic Systems (CEL)). Category B2, D and E are now under discussion, so they are not yet added to the Regulation. However, it is important to know their definition and requirements in order to anticipate future modifications. Below, it is shown the description of the functions added to amend 03:

- Category C: means, a function which is initiated/activated by the driver and which can perform a single lateral manoeuvre (e.g. lane change) when commanded by the driver
- Category D: means a function which is initiated/activated by the driver and which can indicate the possibility of a single lateral manoeuvre (e.g. lane change) but performs that function only following a confirmation by the driver
- Category E: means a function which is initiated/activated by the driver and which can continuously determine the possibility of a manoeuvre (e.g. lane change) and complete these manoeuvres for extended periods without further driver command/confirmation

The development of Regulation 79 is being done in the Informal Group of ACSF under GRVA Group (Figure: 2.3). In it, countries' representatives are not the only participants, but also experts of the sector as vehicles and component manufacturers, electric/electronic experts, etc.

This Regulation becomes relevant to the project because it shows some ADAS functions that are already regulated, and that could be used in order to make a vehicle navigate autonomously in a highway. Especially pertinent are:

- Category B1: Lane Keeping
- Category C: Lane Change

Selecting these two functions would allow our vehicle to perform lateral movements and also to keep the vehicle in the lane.

The lateral movement of the vehicle can be achieved by performing lane changes. Nonetheless, the longitudinal movement can be achieved safely by an acceleration action combined with and the ADAS function: Adaptive Cruise Control. It automatically adjusts the velocity of the vehicle to the vehicle ahead in order to maintain a safety distance.

This function is currently not regulated under any UNECE Regulation. However, is recently being discussed in GRVA Working Party if a new regulation should be created [9].

2.3 Regulations' future

This section will review how UNECE and EC are addressing the irruption of AV, in order to foresee future changes in the regulations.

UNECE, as it has been shown above (Figure 2.2), has created the Working Party on Automated/Autonomous and Connected Vehicles to address WP29 to address AV. The group is divided three Informal Groups and two Task Forces (Figure 2.3). Informal Groups are working on incorporating ADAS functions in old Regulations or creating news for a specific function.
However, lately, with the pressure of old and new manufactures developing AV of SAE level 4/5, UNECE has decided that, instead of regulating function by function, they would create a new Regulation that would set the requirements to approve AV. Despite this will, the creation of this Regulation is technically complex, due to the evaluation of not only mechanical systems, but also software development, AI algorithms, hardware integration, functional safety, cyber security, etc. Therefore, the implementation of such Regulation will take, presumably, many years to come to light.
On the other hand, the EC is also working on the development of AV. In April 2018, EU Transport Ministers signed the Declaration of Amsterdam, where all parts agreed on the next necessary steps in the development of self-driving technologies. These actions were grouped into main four pillars:

- Development of a shared European strategy on connected and automated driving while strengthening the links between existing platforms such as Cooperative Intelligent Transport Systems Platform (C-ITS), Gear 2030 and the Round Table on Connected and Automated Driving

- Continuation of the C-ITS platform for the deployment of interoperable C-ITS in the EU. This would mean also to widen its scope in order to include infrastructure related aspects, traffic management and road safety

- Reviewing of the EU Regulatory framework to support the development and use of automated and connected driving

- Development of a coordinated approach towards research and innovation activities in the field of connected and automated driving

As it can be seen, the third pillar is intended to allow the development of AV SAE level 4/5. In order to help this development, there has been created some expert groups that are attending to UNECE session to assess the development of future Regulations. These groups are present in the already presented GRVA Working Party.

Although EC is seeking to help AV, they are also encouraging the regulation and implementation of ADAS in order to guarantee road safety during the transition years [10].
In conclusion, it is important to know and understand how the regulatory bodies are structured and how they are not only addressing the future of AV, but also the current ADAS functions. If the development of an AV wants to be commercialized in a not far future, it is essential to know the Regulations' requirements.

3 SIMULATORS

3.1 INTRODUCTION

The choice of the adequate environment of simulation is a key step, not only for the development phase of the project, but also in the problem definition. The simulator will determine the measure you get from your model, and therefore how the problem definition is outlined. Nowadays, there exist plenty of simulators in almost all the fields of robotics, due its importance in the development of the control algorithms.

AV's simulator is a challenge which many companies are working on. It includes a wide range of study's fields, not only of Machine Learning (ML), but also of vehicle's dynamics, traffic simulation, etc. This is the reason why there are so many different simulators, where each one is focused on a specific area.

Besides, the use of a simulator is a must in the RL field. It will be explained in the RL section below (4.2), but the main idea is that the need of multiple iterations in order to reach a solution, in some scenarios, would have on one hand a high price if real objects are used and, in the other hand, could lead to dangerous situations (e.g. to make an untrained vehicle travel in a public highway with other drivers).

The following section is centred on the study and comparison of ten simulators in order to, at the end, be able to choose the most adequate for the problem. To do so, it has been selected some specific characteristics which may have more impact on the development of the project:

- General information: The main information about the company/start-up/user responsible of the simulator and repository website
- Description: A description of the simulator and the purpose of this development by the author
- ML integration: How the simulator can integrate or if it is designed to support ML's algorithms/libraries
- Simulator information:

  - Scenarios: which are the scenarios of the simulator (urban road, highway, city, with other vehicles, with pedestrians, etc.)
  - Functions: which are the functions the simulator gives the user to control the vehicle
  - Sensing measurements: which is the data from the vehicle or the environment the simulator gives the user (LIDAR, Radar, Odometry, Cameras, other sensors, position, velocity, etc.)
  - Simulator controls: how can the simulator be controlled (reset environment, accelerate time, control number of steps, etc.)

- Software information:
  - Maturity stage: Is the software still under development? In which version is it?
  - Open source: Is there a license of use
  - Code language
  - Applications

- Software evaluation: five software evaluation factors has been used in order to determine the level of difficulty to operate:
  - Understandability: Software understandability is one of the most important characteristics of software quality because it can influence the cost or reliability at software evolution in reuse or maintenance
  - Documentation: It will cover if there is software Documentation and if it explains how to operate code or how to use the simulator. Besides, it will be evaluated if it is comprehensive, appropriate and well-structured (this evaluation depends on the level and experience of the user). Besides, it is checked the community support
  - Installation: Software and hardware requirements, and how straightforward is the installation in a supported system
  - Learnability: Looking at the documentation, how difficult is to learn how to write your own code on it
  - Portability: Which is the level of difficulty to work with the same project with different computers

Finally, as conclusion for each of the simulators, some advantages and drawbacks are given.

3.2 Simulators' State-of-the-Art

This section will show the state-of-the-art AV simulators. As it has been said before, some characteristics will be shown in order to have a better understanding of the simulator and to evaluate which fits better with our project scope. However, in some of the simulators, some of the descriptions have not been found.

The first criteria to search was to look for simulators that allows the user to control an autonomous vehicle on a road. These were the results:

- AirSim
  - General information: AirSim is a simulator for drones, cars and more built on Unreal Engine. Open source for autonomous vehicles from Microsoft AI & Research, which offers physically and visually realistic simulations
  - Repository/Web: https://github.com/Microsoft/AirSim
  - Description: AirSim is an open-sourced system designed to train autonomous systems. AirSim provides realistic environments, vehicle dynamics, and multi-modal sensing for researchers building autonomous vehicles that use AI to enhance their safe operation in the open world
  - ML integration: The simulator is a platform for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles

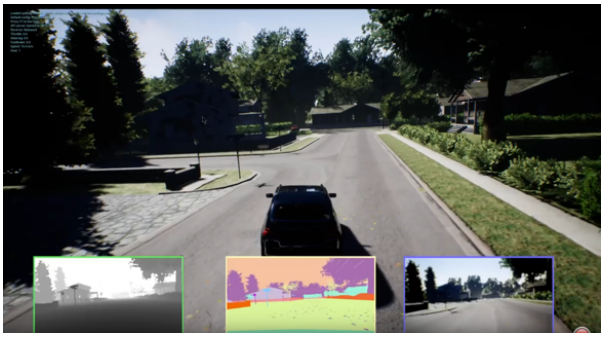

Figure 3.1: AirSim simulator

  - Simulator information:

        * Scenarios: AirSim comes with a detailed 3D urban environment that includes a variety of diverse conditions, including traffic lights, parks, lakes and construction sites. It includes an open world (The simulation contains more than 12 kilometers of drivable roads spanning more than 20 city blocks), realistic environments, multi-vehicles, etc.
        * Functions: Basic control of a vehicle functions. Not ADAS implemented.
        * Sensing measurements: Pose and images. IMU, LIDAR (under implementation)
        * Simulator controls: Creation of multiple vehicles/drones under the control of the user. It can be used with manual controls.
    - Software information:
        * Maturity stage: Under development. Since November 13, 2017 include AV
        * Code language: C++, Python and Java. ROS compatibility.
        * Open source: Yes
        * Applications: Autonomous navigation in a real world scenarios and Dron control.
    - Software evaluation:
        * Understandability: Complex
        * Documentation: Detailed documentation available with community support
        * Installation: Windows and Linux. High computer capacities needed
        * Learnability: Availability of tutorials
        * Portability: Github repository, but installation of softwares is needed. High computational resources needed

- Apollo
    - General information: Apollo, created by Baidu, is a high performance, flexible architecture which accelerates the development, testing, and deployment of Autonomous Vehicle.
    - Repository/Web: https://github.com/apolloauto
    - Description: Apollo provides an open, reliable and secure software platform for its partners to develop their own autonomous driving systems through on-vehicle and hardware platforms

- ML integration: End-to-end Deep-learning algorithms supported

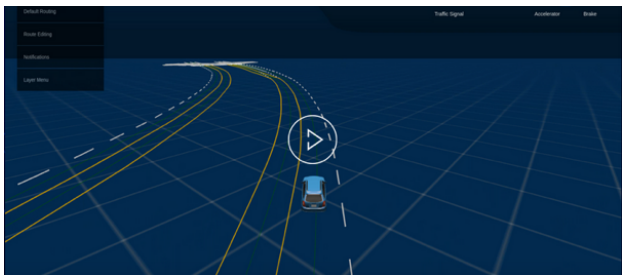

Figure 3.2: Apollo simulator

- Simulator information:
  - Scenarios: The simulation platform allows users to input different road types, obstacles, driving plans, and traffic light states. Developers can create realistic scenarios that support the verification of multiple modules such as perception, and planning, plus traffic flows that provide rigorous testing for algorithms
  - Functions: Gives users a complete setup to run multiple scenarios, and upload and verify modules in the Apollo environment
  - Sensing measurements: Laser Point Cloud Obstacle Detection And Classification, Traffic Light Detection, Image-Based Obstacle Detection And Classification, etc.
  - Simulator controls:
- Software information:
  - Maturity stage: Under development. 3rd July 2018 Apollo 3.0
  - Code language: C++
  - Open source: Yes
  - Applications: Simulation and real world testing
- Software evaluation:
  - Understandability: Complex
  - Documentation: Detailed documentation available with community support
  - Installation: Ubuntu 14.04. High computer capacities needed
  - Learnability: Tutorials and Developer Center Lessons

    * Portability: Github repository, but installation of softwares is needed. High computational resources needed

- Autoware
  - General information: Autoware is the world's first "all-in-one" open-source software for self-driving vehicles. It is ROS-based open-source software, enabling self-driving mobility to be deployed in open city areas
  - Repository/Web:
  - Description: The capabilities of Autoware are primarily well-suited for urban cities, but highways, freeways, mesomountaineous regions, and geofenced areas can be also covered It is provided a ROSBAG-based simulation environment for those who do not own real autonomous vehicles.
  - ML integration: Deep learning algorithms implemented for cameras and LIDAR and available to be implemented in control areas

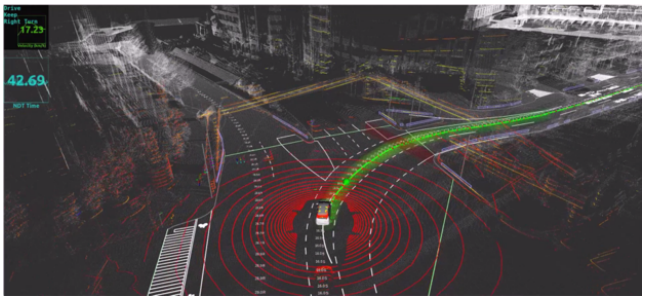

Figure 3.3: Autoware simulator

  - Simulator information:
    * Scenarios: Open city areas
    * Functions: Velocity and angular velocity
    * Sensing measurements: Cameras and LIDAR with sensor fusion algorithms
    * Simulator controls:
  - Software information:
    * Maturity stage: Under development. First version in 2015
    * Code language: C++ and ROS compatibility
    * Open source: Yes

    - Applications: Autonomous vehicle research

  - Software evaluation:

    - Understandability: Complex
    - Documentation: Detailed documentation available with community support
    - Installation: High computer capacities needed
    - Learnability: Free manuals
    - Portability: Autoware Online. You may test Autoware at Autoware Online. No need to install the Autoware repository to your local environment

- Carla

  - General information: CARLA has been developed from the ground up by Intel Labs, Toyota Research Institute and CVC Barcelona, to support development, training, and validation of autonomous urban driving systems
  - Repository/Web: https://github.com/carla-simulator/carla
  - Description: CARLA has been built for flexibility and realism in the rendering and physics simulation. It is implemented as an open-source layer over Unreal Engine 4
  - ML integration: Classic modular pipeline, deep network trained end-to-end via imitation learning, deep network trained via reinforcement learning.

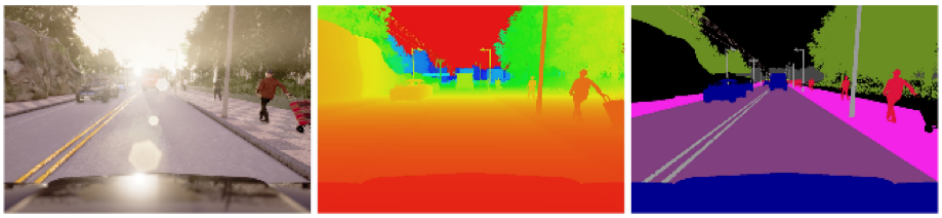

Figure 3.4: Carla simulator

  - Simulator information:

    - Scenarios: The environment is composed of 3D models of static objects such as buildings, vegetation, traffic signs, and infrastructure, as well as dynamic objects such as vehicles and pedestrians. Implemented a basic controller that governs non-player vehicle behaviour. There are implemented two towns.

        * Functions: Commands to control the vehicle including steering, accelerating, and braking
        * Sensing measurements: Sensors are limited to RGB cameras and to pseudo-sensors that provide ground-truth depth and semantic segmentation
        * Simulator controls: The simulator can reset the simulation, change the properties of the environment and modify the sensor suite
    - Software information:
        * Maturity stage: Carla was created in 2016. 30th July 2018 CARLA version 0.9.0 was released: Multi-client multi-agent support
        * Code language: Python
        * Open source: Yes
        * Applications: Free AV navigation
    - Software evaluation:
        * Understandability: Complex
        * Documentation: Detailed documentation available with community support
        * Installation: High computer capacities needed
        * Learnability: Tutorial available
        * Portability: Github repository, but installation of softwares is needed. High computational resources needed

- DeepTraffic
    - General information: AV Simulator created for the competition organized by the MIT in one of its courses
    - Repository/Web: https://selfdrivingcars.mit.edu/deeptraffic/
    - Description: Driving Fast through Dense Traffic with Deep Reinforcement Learning
    - ML integration: Control algorithm based on Deep Reinforcement Learning. Study what hyperparameters have a significant impact on evaluated agent performance and what come at a great cost without much performance gain.
    - Simulator information:

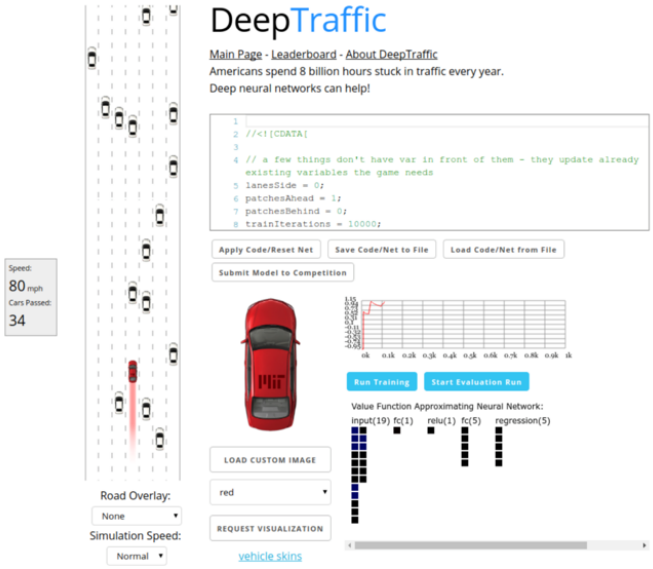

Figure 3.5: DeepTraffic simulator

- Scenarios: Seven path highway, with other cars traveling randomly
- Functions: Controllability of the vehicle: accelerations, deceleration, left and right steering, and do nothing
- Sensing measurements: LIDAR
- Simulator controls: Reset the simulator. Control more than one vehicle.

– Software information:

- Maturity stage: Simulator finished in 2017, but the competition is still open
- Code language: Java Script
- Open source: No
- Applications: AV Navigation in a highway

– Software evaluation:

- Understandability: Easy
- Documentation: Github wiki
- Installation: Not required. Only webside
- Learnability: Easy. Well explained
- Portability: Browser workspace

- Udacity AV Simulator
  - General information: Simulator created by Udacity for a project of its Self-Driving Car Nanodegree
  - Repository/Web: https://github.com/udacity/self-driving-car-sim
  - Description: This simulator was built for Udacity's Self-Driving Car Nanodegree, to teach students how to train cars how to navigate road courses using deep learning. It is intended to use deep neural networks and convolutional neural networks to clone driving behavior
  - ML integration: Control of the vehicle build to implement a Deep Learning algorithm

Figure 3.6: Udacity AV Simulator

  - Simulator information:
    - Scenarios: Two path track. It can be modified and created as to the users will
    - Functions: The model will output the steering angle of the vehicle
    - Sensing measurements: Image data and steering angles
    - Simulator controls: Availability to create your own track
  - Software information:
    - Maturity stage: Finished. Last version: Version 2, 2/07/17
    - Code language: C
    - Open source: No
    - Applications: Learning by demonstration control for an AV

  - Software evaluation:
    * Understandability: Complex
    * Documentation: Available
    * Installation: Linux, Mac and Windows. Unity based.
    * Learnability: Explanation coursing Udacity's Self-Driving Car Nanodegree
    * Portability: Github repository, but installation of softwares is needed

- Udacity Highway-path-planning
  - General information: Simulator created by Udacity for a project of its Self-Driving Car Nanodegree
  - Repository/Web: https://github.com/mithi/highway-path-planning
  - Description: The simulator was built for Udacity's Self-Driving Car Nanodegree, to teach students how to create a path planning pipeline that would smartly, safely, and comfortably navigate a virtual car around a virtual highway with other traffic
  - ML integration: Control the vehicle using classic control techniques. Not first considered to be used with ML techniques

Figure 3.7: UDACITY Highway-path-planning simulator

  - Simulator information:
    * Scenarios: Six way highway with two directions. Other cars traveling in the highway performing random movements

        * Functions: Give to the car next point to visit
        * Sensing measurements: Car's localization's and sensor data fusion for the surroundings
        * Simulator controls: Not available to reset the simulator
    - Software information:
        * Maturity stage: Finished. Last version: 1/08/17
        * Code language: C
        * Open source: No
        * Applications: Classic control for an AV
    - Software evaluation:
        * Understandability: Complex
        * Documentation: Available
        * Installation: Linux, Mac and Windows. Unity based.
        * Learnability: Explanation coursing Udacity's Self-Driving Car Nanodegree
        * Portability: Github repository, but installation of softwares is needed

- Robotbenchmark
    - General information: Robotbenchmark offers a series of robot programming challenges that address various topics across a wide range of difficulty levels. These benchmarks are provided for free as online simulations, based on a 100% free open source software stack. The performance achieved by users is recorded and displayed online
    - Repository/Web: https://robotbenchmark.net
    - Description: Program a Lincoln MKZ autonomous car to drive as fast as possible on a crowded highway
    - ML integration: Availability to import Python Libraries
    - Simulator information:
        * Scenarios: Straight highway with four lanes (including one emergency lane)
        * Functions:

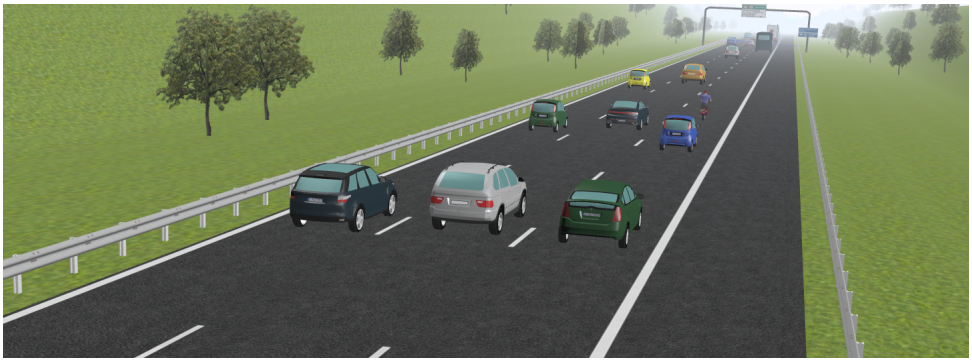

Figure 3.8: Robotbenchmark simulator

    - Sensing measurements: Radar and camera with already computer vision recognition's methods implemented
    - Simulator controls: Not allowed simulation restart in the free version
  - Software information:
    - Maturity stage: Finished
    - Code language: Python
    - Open source: Yes
    - Applications: Testing of take-over control algorithms in vehicles
  - Software evaluation:
    - Understandability: Moderated
    - Documentation: Available
    - Installation: Not required
    - Learnability: Simple examples are given in order to understand this simulator
    - Portability: Easy to use in other computer due it works in the browser

- Metacar
  - General information: A RL environment for self-driving cars in the browser created by Thibault Neveu
  - Repository/Web: https://github.com/thibo73800/metacar
  - Description:

- ML integration: Environment made to test RL algorithms. Tensorflow.js library implemented in the already implemented control algorithms

Figure 3.9: Metacar simulator

- Simulator information:
  - Scenarios: Urban road with two path with two directions. The library let you create your own levels and personalize the environment to create your desired scenario
  - Functions: control of steering and velocity of the vehicle: accelerate, decelerate, steer left or right, and wait
  - Sensing measurements: Front LIDAR. Collisions with other vehicles. Detection of the vehicles, ground, and road. Detection of the car going out track.
  - Simulator controls: Environment similar to Open GYM simulators. It can be played, stopped, reset, but not accelerated
- Software information:
  - Maturity stage: Under development. The project started in June 2018. Last modification November 2018.
  - Code language: Java and HTML
  - Open source: Yes
  - Applications: Testing of RL algorithm on an AV
- Software evaluation:
  - Understandability: Easy
  - Documentation: Not available

    * Installation: Not required
    * Learnability: Not tutorials available, however some examples are provided
    * Portability: Easy to implement in other computers due to its browser use

- Unity ML-agents Highway Simulator
  - General information: Simulator created by Kyushik Min, based on Unity ML-Agents, in order to test RL algorithms of AV navigation on a highway
  - Repository/Web: https://github.com/MLJejuCamp2017/DRL_based_SelfDrivingCarControl
  - Description: The simulator reproduces the behaviour of a crowded highway
  - ML integration: The project was created to control the vehicle with RL algorithms

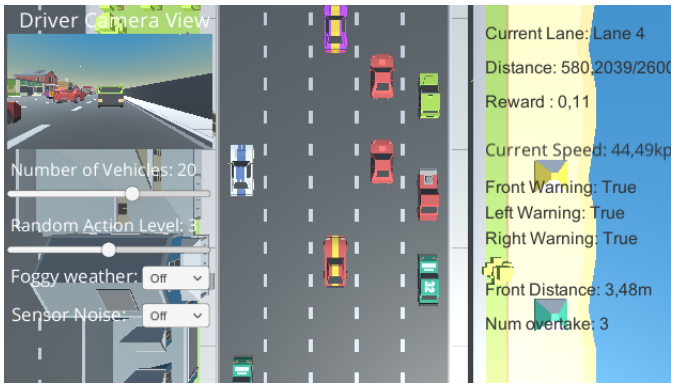

Figure 3.10: Unity ML-agents Highway simulator

  - Simulator information:
    * Scenarios: Straight highway with five lanes of the same direction
    * Functions: Right and left lane change, accelerate, decelerate and wait
    * Sensing measurements: Camera and 360 degrees LIDAR
    * Simulator controls: Environment similar to Open GYM simulators. It can be played, stopped, reset, but not accelerated
  - Software information:
    * Maturity stage: Finished
    * Code language: Python
    * Open source: Yes

        * Applications: testing of RL algorithms in a AV on a highway
    - Software evaluation:
        * Understandability: Moderated
        * Documentation: Poor
        * Installation: Linux, Mac and Windows. Unity based. Depending on the applications it does not demand a computer with high capacities
        * Learnability: Examples are provided
        * Portability: Installation is required, but may be installed in almost any computer

3.3 SIMULATORS' COMPARISON

In this section, taking into account the characteristics of each simulator presented above, a comparison will be done. Then, at the end, one will be selected and implemented to develop the project.

In order to compare all the simulators, the previous characteristics selected to describe each simulator have been grouped in the following comparative elements:

- Project objectives (Table: 3.1): In order to evaluate the alignment between the project objectives and scope with the simulator characteristics, the following elements were taken into account:
    - ML integration: Due to the objective of the project is the implementation of a RL in a simulator, the case where the simulator was intended to implement ML methods gains more attention. Where YES if it was intended to implement ML, and otherwise NO
    - Simulator information: Either the scenarios or the functions or the sensing data or the controls are important factors at the moment to pick the simulator. All will play an important role in the development. Thus, the closer to our project approach the better. Where YES if the scenarios, sensors or controls fulfil our project's initial approach, APPROX if there are some elements that fulfils and others that do not and NO if they do not fulfill
- Implementation (Table 3.2): In order to evaluate the degree of difficulty of the implementation of the simulator in our environment, the following elements were taken into account:

Project Objectives				
Simulators	**ML integration**	**Simulator information**		
		Scenarios	**Sensor**	**Control**
AirSim	YES	APPROX.	YES	YES
Apollo	YES	APPROX.	YES	YES
Autoware	YES	APPROX.	YES	YES
Carla	YES	APPROX.	YES	YES
DeepTraffic	YES	YES	YES	YES
Udacity AV Simulator	NO	APPROX.	NO	YES
Udacity Highway-path-planning	NO	YES	NO	NO
Robot-benchmark	YES	YES	YES	YES
Metacar	YES	YES	YES	YES
Unity ML-Agents Highway Simulator	YES	YES	YES	YES

Table 3.1: Simulators' comparison: Project Objectives

- Software information: The maturity stage of the project and code language are important factors in order to evaluate if it is feasible to implement the simulator. If the project is in the first stages, it may mean finding unexpected difficulties (EARLY STAGE). If the project is in its final stage, but it is not being updated, it may mean to fins unexpected errors (END). Otherwise, if the project is still under development, but with some stable versions released, could mean to be a good option to be chosen (DEVEL.).

  Code language is also an important factor to take into account, as it will help or not using our prior coding knowledge the implementation of the simulator and the RL algorithm.

- Software evaluation: Understability (EASY-MEDIUM-COMPLEX), Documentation (YES-NO) and Learnability (EASY-MEDIUM-COMPLEX) factors will help to evaluate if the simulator is inside the learning scope of our project

- Hardware requirements: Finally in order to understand if the simulator can be installed and run in our computer, the hardware requirements have been compared:

  - Our computer:

Implementation					
Simulators	**Software information**		**Software evaluation**		
	Maturity	**Language**	**Understability**	**Documentation**	**Learnability**
AirSim	DEVEL.	C++, Python Java, ROS	COMPLEX	YES	COMPLEX
Apollo	DEVEL.	C++	COMPLEX	YES	COMPLEX
Autoware	DEVEL.	C++, ROS	COMPLEX	YES	COMPLEX
Carla	DEVEL.	Python	COMPLEX	YES	COMPLEX
DeepTraffic	DEVEL.	Java	EASY	YES	EASY
Udacity AV Simulator	END	C	COMPLEX	YES	COMPLEX
Udacity Highway-path-planning	END	C	COMPLEX	YES	COMPLEX
Robot-benchmark	DEVEL.	Python	EASY	YES	MEDIUM
Metacar	END	Java, HTML	EASY	NO	MEDIUM
Unity ML-Agents Highway Simulator	END	Python	MEDIUM	YES	MEDIUM

Table 3.2: Simulators' comparison: Implementation

Laptop: Macbook Pro Retina, 13-inch, Early 2015

Processor: 2,7 GHz Intel Core i5

RAM Memory: 8 GB

GPU: Intel Iris Graphics 6100 1536 MB

– Simulators' hardware requirements: Table 3.3

Then, a final comparison table has been created with all the simulators and the comparative elements: Table 3.4. Additionally, a drawback column has been added to the Table 3.4 in order to provide more information. In it, not only conclusion about the table's comparative factors are given, but also personal experience obtained by trying to install and work with these simulators.

Once the comparison tables has been made, it can be seen that for Hardware and implementation complexity, AirSim, Apollo, Autoware and Carla simulators were discarded to be implemented.

DeepTraffic simulator fulfilled most of the requirements to be implemented. However, because it is a closed environment where only hyperparameters, of a Q-Learning algorithm already imple-

Hardware requirements	
Simulators	**Requirements**
AirSim	HIGH
Apollo	HIGH
Autoware	HIGH
Carla	HIGH
DeepTraffic	LOW
Udacity AV Simulator	MEDIUM
Udacity Highway-path-planning	MEDIUM
Robot-benchmark	LOW
Metacar	LOW
Unity ML-Agents Highway Simulator	LOW

Table 3.3: Simulators' comparison: Hardware requirements

mented, could be tweaked, it was discarded as well. Also, due to the competition's popularity, there were already a lot of people developing new algorithms to tweak those hyperparameters.

Udacity AV Simulator and Udacity Highway-path-planning simulator were a good option at the beginning due to its original educational purpose to learn about AV. However, both were addressed to its respective projects' objectives and any change to the environment was very tedious. For this reason, both simulators were discarded.

Robotbenchmark was a promising simulator. It fulfilled almost all the requirements. However, once it was set up, one of the authors, Oliver Michel, explained that in order to save CPU/GPU power on their servers, the simulator could not be rest automatically. In RL is very important to be able to repeat the simulation several times, and if it meant that it had to be done manually, it would become a very tedious and tiring process. This was the reason to discard this simulator.

Metacar simulator was discarded because its lack of documentation. It was a project created by Thibault Neveu which was finished one year ago, and some of the documentation and examples

Simulator	Project objectives	Implementation	Hardware requirements	Drawbacks
AirSim	FULFILS	COMPLEX	NOT FULFILS	It is a powerful simulator, however its complex implementation is out of the scope of the project
Apollo	FULFILS	COMPLEX	NOT FULFILS	It is a powerful simulator, however its complex implementation is out of the scope of the project
Autoware	FULFILS	COMPLEX	NOT FULFILS	It is a powerful simulator, however its complex implementation is out of the scope of the project
Carla	FULFILS	COMPLEX	NOT FULFILS	It is a powerful simulator, however its complex implementation is out of the scope of the project
DeepTraffic	FULFILS	EASY	FULFILS	It only allows the user to tweak the hyperparameters of the RL algorithm, not to implement your own
Udacity AV Simulator	NOT FULFILS	COMPLEX	FULFILS	It does not allow to get the data wanted to develop our project
Udacity Highway-path-planning	NOT FULFILS	COMPLEX	FULFILS	ML algorithms were not considered to be implemented. Therefore, the simulator does not allow to rest the environment whenever you want
Robot-benchmark	FULFILS	MODERATE	FULFILS	It does not allow to reset the environment by software (only manually). For RL algorithms is important to have this possibility in order to automate the training
Metacar	FULFILS	MODERATE	FULFILS	Good simulator created by a PhD student in order to test RL algorithms, but the lack of documentation makes its implementation tedious
Unity ML-Agents Highway Simulator	FULFILS	MODERATE	FULFILS	It fits with project scope, however its lack of documentation may lead to implementation problems

Table 3.4: Simulators' comparison

disappeared. Another reason was coding language: it was implementing TF Library in Java, and our previous knowledge about TF was in Python.

Finally, Unity ML-Agents Highway Simulator was selected to develop the project. It fulfilled all the project's requirements, as well as the hardware's. In the following section (3.4.2), the simulator will be explained, pointing out its advantages and drawbacks. Then, a guideline of its installation will be given.

3.4 Simulator selected

The simulator chosen, as it has been explained before, is the simulator made by Kyushik Min and Hayoung Kim and uploaded on Github: Unity ML-Agents Highway Simulator [18].

From on hand, it has several advantages among the others simulators, but on the other, it present some disadvantages as well. In the following section, an overview of the simulator will be first done. Then the advantages and disadvantages will be presented more in detail and finally, an exhaustive tutorial about the implementation of the simulator in a working environment will be explained.

3.4.1 Simulator's Overview

The simulator consists on a five straight lanes of one direction highway, with a host vehicle (red vehicle) navigating among other vehicles (Figure 3.11). All vehicles of the simulators are cars or bans. However, the host vehicle is always a red car. Thus, not motorcycles, trucks or bicycles are simulated.
All vehicles perform random ADAS's actions (explained below), except the host vehicle, which will be controlled by the user.
The simulator was created using the Unity ML-Agents, which offers all the potential of the Unity environment in order to create the simulator and, at the same time, Unity ML-Agents offers a flexible way to develop and test new AI algorithms quickly and efficiently across a new generation of robotics, games, and beyond [19].

As it has been already stated, the agent vehicle will be navigating among other vehicles which, for the purpose of the project, will be performing different ADAS' functions using a RL algorithm.

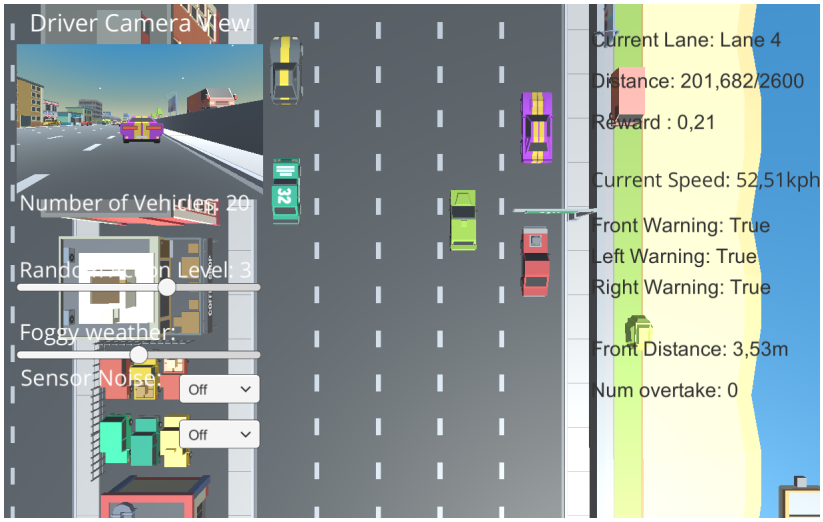

Figure 3.11: Unity ML-Agents simulator

In order to feed the RL algorithm (explained in Section 4.2) with the actual state of the host vehicle, some sensing devices are needed. The sensors implemented in the simulator are:

- RGB camera: Provides RGB vision of the vehicle's forward view (Figure 3.12)

Figure 3.12: Camera's image

- LIDAR: Provides the distance of 360 degrees of the vehicle's surroundings (Figure 3.13)

Besides these sensors' data gotten from the simulator, other data can be obtained:

- Forward distance from the vehicle to the vehicle in front
- Forward vehicle speed
- Host vehicle speed
- Number of overtakes
- Number of Lane Change (LC)

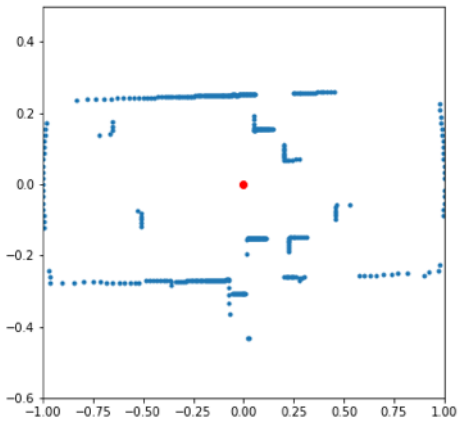

Figure 3.13: LIDAR's plot

As well, the simulator has implemented some ADAS functions which will help us to make the RL algorithm to learn to navigate safely:

- Left warning: It tells if there is a vehicle in the left side of the host vehicle, so it should not perform a left LC. This function corresponds to the commercialized Blind Spot ADAS's function
- Right warning: It does the same as the above warning, but in the right side of the vehicle
- Forward warning: It provides a warning if the distance between the two vehicles is lower than a certain value. This function corresponds to the commercialized Forward Collision Warning System ADAS's function

The actions implemented in the vehicle in order to make it travel through the highway are:

- Accelerate and Lane Keeping: It accelerates the host from the current host vehicle's velocity up to the maximum highway's velocity while keeping the vehicle in the center of its actual lane
- Do nothing and Lane Keeping: It maintains the vehicles at a velocity between the maximum highway's velocity and the minimum safety travel velocity while keeping the vehicle in the center of the lane

- Decelerate and Lane Keeping: It decelerates the host vehicle from the current host vehicle's velocity up to the minimum safety travel velocity while keeping the vehicle in the center of its actual lane
- LC Left: It performs a lane change to the left by steering the host vehicle to the left
- LC Right: It performs a lane change to the right by steering the host vehicle to the right

In addition, another relevant property to the project is that the simulator provides the option of reset the environment: The simulator allows the user to reset the environment using a software command and not manually.

Finally, out of the scope of the project but interesting in order to improve the algorithm, the simulator presents these additional options:

- Noise: Change the noise weight of the LIDAR sensor
- Foggy weather: Add fog to the environment. Thus, the camera will not capture a clear image
- Multi agent: Control not only the agent vehicle, but also other vehicles
- Random action selection: Change the weight of the random actions performed by the other vehicles. More weight more actions taken
- Number of vehicles in the highway: Change the number of vehicles in the highway from 0 to 32 vehicles

In the following section advantages and drawbacks of the simulator are given.

3.4.2 Simulator: Advantages and Disadvantages

Unity ML-Agents Highway Simulator presented some advantages that made take the decision to select this among the other simulators. These advantages were:

- Implementation: The simulator, as it was implemented using Unity engine, it could be run in our Operating System (macOS).
- Code language: The simulator could be interacted using Python files. It was an important advantage due to our previous knowledge about Python and, at the same time, because TF is optimized to be used with this language

- ML oriented: The implementation using Unity ML-Agent made the simulator more convenient to be used to implement and test RL algorithms
- OPEN AI GYM similarity: The similarity of the interaction between the simulator and the control algorithm with OPEN AI GYM environments and the previous knowledge about it, could imply a faster learning

Once the simulators' comparison and selection was done, some other drawbacks of the simulator appeared:

- The documentation about the simulator data acquisition was not totally correct
- The lack of examples complicates the implementation
- The lack of knowledge about how the simulator was created, did not allow to accelerate the training by not showing the simulator interface

Despite these drawbacks, it seemed to be the best fit simulator for our project.
In section 3.5, a guideline of its implementation is given.

3.5 Simulator Implementation's Guideline

The first, the installation of the simulator will be shown. Then, how to create a Python file to interact with the simulator will be explained.

Before starting downloading the simulator and the repository and in order to not have future compatibility problems, our work environment should have:

- Python version 3.6.5
- Anaconda version 5.2.0
- TensorFlow version 1.8.0 (if working with TF)

Also, these Python Libraries should be installed in order to be able to implement algorithms and visualize results (libraries and Software used to implement the project will be explained in Section 5.1):

- Numpy

- Matplotlib

In order not to have incompatibilities with already installed Libraries, a working environment will be created:

```
conda create --name your_env_name python=3.6.5 tensorflow=1.8.0 numpy=1.16.2 matplotlib
    =3.0.3
```

A working environment is a directory that contains a specific collection of Conda packages that you have installed.

Then, once the environment is created, it needs to be activated each time it is wanted to work with it:

```
conda activate your_env_name
```

And, in order to stop using the environment:

```
conda deactivate
```

Then, it is needed to download the Github repository:

```
git clone https://github.com/MLJejuCamp2017/DRL\_based\_SelfDrivingCarControl.git
```

Once the repository has been downloaded, the simulator will be downloaded and placed inside the repository folder: *./DRL_based_SelfDrivingCarControl-master/environment*

Finally, our algorithm file in charge to interact with the environment will be saved in the folder: *./DRL_based_SelfDrivingCarControl-master/RL_algorithms*

Up to here, the simulator has been installed and it is known where to save the control algorithm. Now it will be shown how to interact with the simulator using Python:

Initially, the previous mentioned Libraries will be imported:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
```

Additionally, Unity Environment will be imported in order to allow the interaction with the simulator:

```
from mlagents.envs import UnityEnvironment
```

Once it has been imported, the environment will be launched (it also begins communication with the environment):

```
env = UnityEnvironment(file_name="./environment/jeju_camp.app")
```

When creating the environment, *Brains* (Policy) responsible of controlling the actions of their associated *Agents* (vehicles) will be created as well. Then, using the following command, the *Brain* of the host vehicle will be selected:

```
default_brain = env.brain_names[0]
brain = env.brains[default_brain]
```

Similar to OpenAI GYM, the environment needs to be reset to initialize it and to start receiving information:

```
env_info = env.reset(train_mode=True)[default_brain]
```

The initial position of the host vehicle is in the center lane (3rd lane) with random initialization of the other vehicles.

env_info will store the information received from the environment. It will be stored in two variables:

- env_info.visual_observations: will be a vector responsible to store the image information. The following function may be used to plot the camera image (Figure 3.12 shows a camera frame):

```
    Num_obs = len(env_info.visual_observations)

    def mostrar_imatge(Num_obs, photo):
        if Num_obs > 1:
            f, axarr = plt.subplots(1, Num_obs, figsize=(20,10))
            for i, observation in enumerate(photo):
                if observation.shape[3] == 3:
                    axarr[i].imshow(observation[0,:,:,:])
                    axarr[i].axis('off')
                else:
                    axarr[i].imshow(observation[0,:,:,0])
                    axarr[i].axis('off')
        else:
            f, axarr = plt.subplots(1, Num_obs)
            for i, observation in enumerate(photo):
```

```
                if observation.shape[3] == 3:
                    axarr.imshow(observation[0,:,:,:])
                    axarr.axis('off')
                else:
                    axarr.imshow(observation[0,:,:,0])
                    axarr.axis('off')

```

However, as in the project only LIDAR data will be used, previous function will not be used.

- env_info.vector_observations: will be the vector responsible to store the following data:
  - glslidar data: from vector position 0 to 359 (env_info.vector_observations[0, 0:359]). It will be a vector of 360 positions corresponding to 360 degrees. The distance is normalized. In order plot the LIDAR data (Figure 3.13 shows a LIDAR plot), the following conversion is done in order to plot the date into coordinates x, y:

```
x = env\_info.vector\_observations[0,0:-14]*np.cos(graus\_1*np.pi/180)
y = env\_info.vector\_observations[0,0:-14]*np.sin(graus\_1*np.pi/180)
```

    LIDAR is positioned looking forwards. Therefore, the 0 degree corresponds to the distance of between the vehicle and an obstacle in front. The Figure 3.14 shows the vector position of the LIDAR degrees.

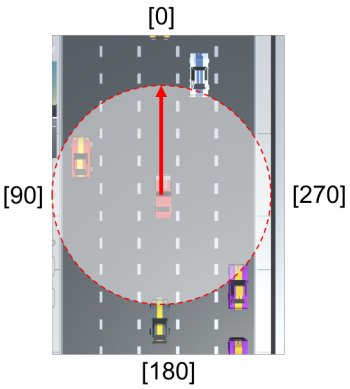

Figure 3.14: LIDAR vector's position

  - Warnings: Left warning, Right warning and Forward warning correspond to positions 360 to 362
  - Forward distance to the vehicle: Normalized distance between the host vehicle and the vehicle in front (vector position 363)

- Forward vehicle speed: vector position 364
- Host vehicle speed: vector position 365
- Number of LC: vector position 366
- Number of overtakes: vector position 367

The *brain* will also tell the action space of the Agent:

```
    Num_action = brain.vector_action_space_size[0]
```

Each of the actions corresponds to a number, which will be interpreted by the Agent to perform the movement. The actions, as it was stated in Section 3.4.1, are:

- Accelerate and Lane Keeping: 3
- Do nothing and Lane Keeping: 2
- Decelerate and Lane Keeping: 4
- LC Left: 0
- LC Right: 1

Then, in order to send the action to the agent, the *step* function needs to be executed:

```
    env_info = env.step(action_number)[default_brain]
```

With this information, a control algorithm can already be implemented. The following code provides a very simple example of a hard-coded control algorithm using only some LIDAR information during 500 steps:

```
# SelfDriving Car -- TESTS

import numpy as np
from mlagents.envs import UnityEnvironment

env_name = "../environment/jeju_camp.app"
train_mode = True
env = UnityEnvironment(file_name=env_name)

default_brain = env.brain_names[0]
brain = env.brains[default_brain]

```

```
env_info = env.reset(train_mode=train_mode)[default_brain]

Num_action = brain.vector_action_space_size[0]
action = np.zeros([Num_action])
action = 1
steps_total = 500

for iteration in range(steps_total):
    print (iteration)

    if iteration%1 == 0:
        env_info = env.step(4)[default_brain]

    if env_info.vector_observations[0,0] < 0.3:
        #print ("forward warning")
        if env_info.vector_observations[0,27] > 0.2:
            #print ("Left steer")
            action = 0
        elif env_info.vector_observations[0,337] > 0.2:
            #print ("Right steer")
            action = 1
        else:
            #print("decelerating")
            action = 2
    else:
        #print("accelerating")
        action = 3
```

4 Reinforcement Learning

4.1 Introduction

Reinforcement learning is an approach of ML and, therefore, a branch of AI. It is different to the other ML types due to its objective is to learn a behaviour based on an environment.

Figure 4.1 shows a basic schema of a RL setting. The main characters are:

- Agent: is the main actor. It is the learning system responsible of observing the environment, select and perform Actions
- Policy: is the "brain" of the Agent. Is the strategy responsible for choosing the best Actions to get the maximum Reward based on the current state
- Action: is a set of possibles moves the Actor can perform in the environment
- Reward: is the reward or penalty (negative reward) gotten from the Environment after performing Actions. It is the feedback telling the success or the failure of the Agent's Action
- Environment: is the physical world where the Agent moves
- State (Observation): is the current and concrete situation of the Agent in the environment

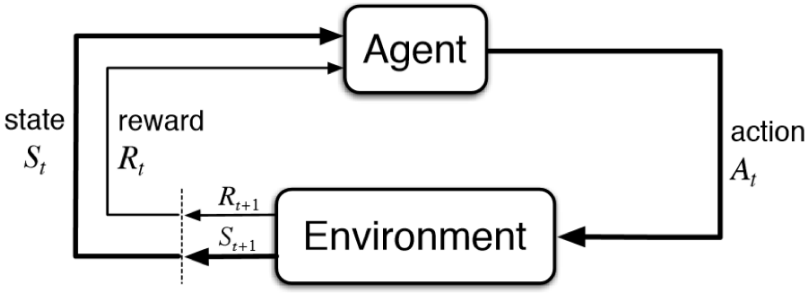

Figure 4.1: Reinforcement learning schema

In summary, the Agent observes the Environment, selects and performs Actions, and gets Rewards. Then, in order to get most Reward over time, the Policy learns by itself what is the best strategy by defining which are the Actions the Agent should choose when it is in a given State.

One of the challenges RL presents is the need of a working environment to train the agent. As many iterations will be required to achieve a good result, the best approach is to use a simulator. Otherwise, if you want to work with a real environment, in some cases, it is impossible to recreate, and another, it would require a high budget and much more time (Simulator advantages are explained in the above Section 3.1).

During the project development, and in order to learn more about RL and its implementations, OpenAI GYM simulators have been used to test and visualize our own implementations.

In the recent years, great success has been achieved using RL techniques. For instance, in 2013, DeepMind startup demonstrated that their RL algorithm could play any Atary game. And three years after, it beat the world champion of the game *Go* [11].

Despite all this great success in this field of research, in AV's world there are not already implemented RL algorithm to control the vehicle. According to Lex Fridman in their MIT 6.S091 Lecture [12], companies are using AI specially in the field of perception, but not in the control part. However, in a more research field, there has been tests using different algorithms. The two main algorithms used are:

- Policy Gradient
- Q-Learning

In this project, Policy Gradient method will be, first, studied and explained and then, implemented. The first implementation will be done in a GYM environment due to its facility on implementation, and then, in the development section (5) it will be implemented together with the AV simulator.

4.2 Policy Gradient

Policy Gradient is one of the most important techniques in RL. During the pasts years, it has become one of the main techniques with more success in ML's category. In order to have a better understanding of this method, a brief explanation will be given:

As it has been explained before in the section 4.1, the general schema of RL is:

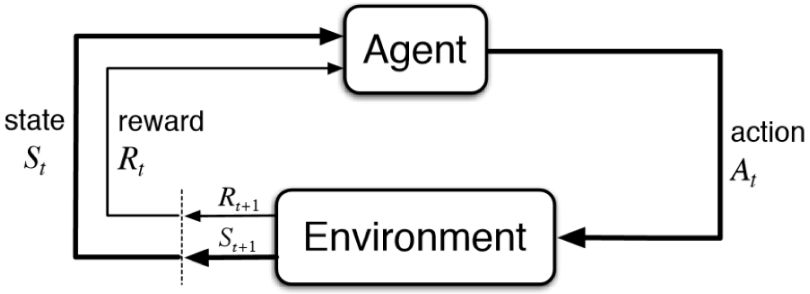

Figure 4.2: Reinforcement learning schema

Where the agent through a policy takes some actions within the environment. Then it receives from the environment, the reward and the observations of the state.

The goal of PG is to find a policy which given some states (inputs) and over some actions (outputs) is able to maximize the expected sum of rewards [15].

The Policy will be a neural network (Figure: 4.3) that processes the state information through some layers of neurons and ends up with a distribution over all possible actions that you might want to take. Then, from this distribution it is sampled an action which is the action that will be taken by the agent. Finally, new rewards and states are gotten. This process is repeated until you end with the episode.

The policy will be stochastic, due to it allows to do exploration and exploitation, in order not to follow always the same path. This means that at the moment the agent has to choose which action to take, it will not always chose the one with more probability but it will have a factor of randomness. The policy will be:

$$\pi_\theta(u, s) \tag{4.1}$$

Where:

- u: actions
- s: states

As explained above, the goal is to maximize the expected rewards. Then the function to maximize will be:

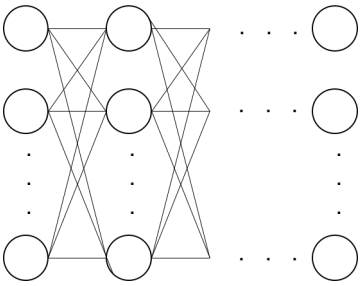

Figure 4.3: Policy

$$\max_{\theta} \mathbb{E}\left[\sum_{t=o}^{H} R(s_t)|\pi_\theta\right] \tag{4.2}$$

Where:

- θ: parameters of the policy
- R: reward

The reward can also be expressed in relation with a sequence state-action, which is the sum of all expected rewards along the trajectory:

$$R(\tau) = \sum_{t=0}^{H} R(s_t, u_t) \tag{4.3}$$

Where:

- τ: is the state-action sequence $(s_o, u_o, ..., s_H, u_H)$

So the utility function of the policy with the parameter vector θ is the expected of rewards:

$$U(\theta) = \mathbb{E}\left[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta\right] \tag{4.4}$$

And compacting the expression using the sequence state-action τ, the utility expression is:

$$U(\theta) = \sum_{\tau} P(\tau;\theta) R(\tau) \tag{4.5}$$

This is the sum over all possible trajectories, probability of that trajectory under the policy multiplied by the reward associated with that trajectory. In other words, is the general measure of the events that can happened in the environment; how likely is the next state given the current state

and the current action.

What is wanted is to maximize the utility function:

$$\max_{\theta} U(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau) \tag{4.6}$$

In order to maximize the previous formula, the gradients need to be computed:

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_{\tau} P(\tau;\theta)R(\tau) \tag{4.7}$$

$$= \sum_{\tau} \nabla_\theta P(\tau;\theta)R(\tau) \tag{4.8}$$

$$= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \tag{4.9}$$

$$= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \tag{4.10}$$

$$= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta)R(\tau) \tag{4.11}$$

With this last formula, it can be seen that it is the sum of trajectories (τ) weighted by probability under to current policy which is an expectation. This expectation can be approximated to:

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)R(\tau^{(i)}) \tag{4.12}$$

This approximation is the empirical estimation for m samples path under the policy π_θ.

Once this formula has been found, the probability of a path $P(\tau;\theta)$ can be expressed as:

$$\log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \tag{4.13}$$

Where the probability of a path is the product of probabilities of the next state given the current state and current action, multiplied by the probabilities of an action given a state under the policy.

It can be simplified as:

$$\nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \tag{4.14}$$

$$\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \tag{4.15}$$

From this equation, it can be seen that the gradient may be computed having only access to our policy.

Replacing this simplified equation in the derivative of the utility equation (4.12), it is obtained:

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) R(\tau^{(i)}) \tag{4.16}$$

In conclusion, it is wanted to obtain the gradients in order to maximize the utility function. By doing so, we will increase the probability of the path with positive rewards and decrease the probability of the path with negative reward. But there might be the case where, for example, the reward is always positive. Then both probabilities would be increased; therefore, it would take many more samples to find the optimum solution.

In order to reduce the variance (the need of high number of samples) a new concept was introduced in 1992 by Ronald J. Williams [17]: baseline.

The main purpose of the baseline is to decrease the variance by increasing the probability of paths that are better in average and decreasing the probability of those that are worst on average.

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) (\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - b(s_k^{(i)})) \tag{4.17}$$

It was also demonstrated [17] that by introducing the baseline, the result will not be biased as long as this number was not directly dependent to an action.

The previous equation (4.17) represents that the gradient of the log probability of an action taken

will be increased, given a state and by the rewards experienced from that time onwards, if it is better of what it would be gotten on average from that time onwards. And if it is worst than average probability would this probability will be decreased.

There are different types of baselines:

- Constant baseline: computing the average of the rewards $b = \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)})$
- Optimal constant baseline [17]
- Time-dependent baseline
- State-dependent expected return

The last type of baseline, State-dependent expected return, is based only on the state (not on the action), and tells how much rewards the agent is going to get from the state onwards.
In this case the baseline is equal to a value function:

$$b = V^{\pi}(s_K^{(i)}) \tag{4.18}$$

Thus:

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - V^{\pi}(s_K^{(i)})) \tag{4.19}$$

In order estimate the value of the reward, a separate neural network will be implemented (Figure: 4.4). This will be an optimization problem using supervised learning, due to, at the end of a batch, we will know the real discounted reward. Therefore, the loss function of this neural network will be MSE (Mean Square Error):

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(R_t - V^{\pi}(s_K^{(i)}))^2 \tag{4.20}$$

In summary, in order to reduce the high variance once the Vanilla algorithm has been implemented is to introduce the baseline. But there exist another way to reduce even more the variance: function approximation.

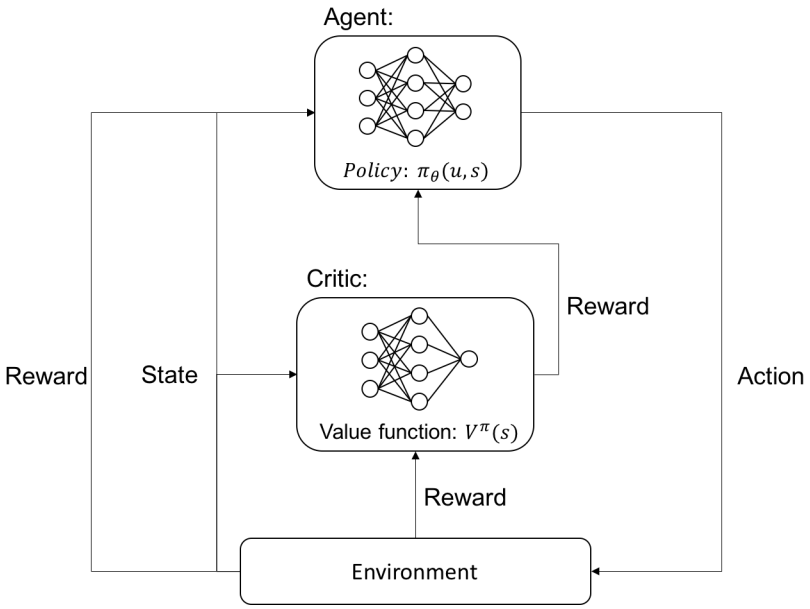

Figure 4.4: Actor-Critic scheme

Implementing this function approximation means to replace the sum of all the rewards by the reward we get immediately plus the value a number of future rewards plus the value estimated by the value function (Equation: 4.21).

$$R_t = Q^{\pi,\gamma}(s,u) = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 V^\pi(s_3)] \tag{4.21}$$

Finally, the utility function, reducing the variance introducing the baseline and the function approximation, is:

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log\pi_\theta(u_t^{(k)}|s_t^{(k)})(\hat{Q}_i(s_t^{(k)},u_t^{(k)}) - V_{\phi i}^\pi(s_t^{(k)})) \tag{4.22}$$

With these gradients computed, it is needed to apply the gradient descend to update all the weights and biases of the neural network:

$$\theta_{i+1} = \theta_i + \alpha\nabla_\theta U(\theta) \tag{4.23}$$

Where:

- α: learning rate

Figure 4.5 shows an implementation of the previous algorithm in a simplified form.

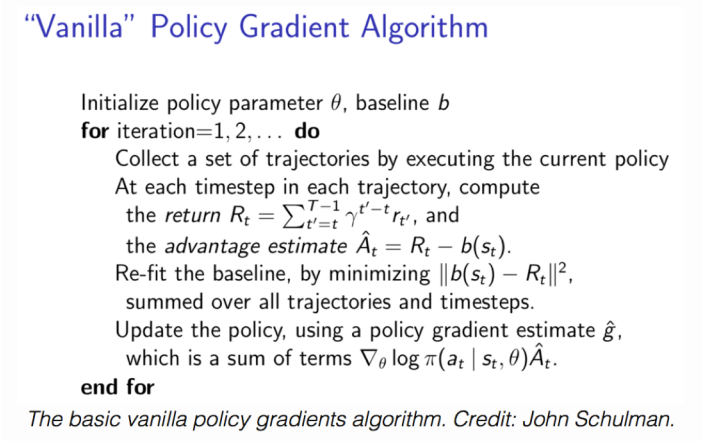

Figure 4.5: Policy Gradient "Vanilla" algorithm

4.2.1 Policy Gradient: Advantages and Disadvantages

In this section a review of the advantages and disadvantages of the PG will be given, in order to know which issues may be found in the implementation problem, as well as, how to improve the model in future work:

- Advantages
  - Convergence: PG method uses the gradients of the loss functions in order to tweak the weights and converge to an optimal maximum. By following those gradient it is guaranteed to find a local or global maximum
  - Easy implementation: setting up a PG algorithm it may be easier than other methods
  - High dimensional action spaces: PG are more effective in high dimensional action spaces than other RL methods as Q-Learning. Q-Learning needs to compute the output for each possible action, which in a high dimensional space would take a lot of time
  - Stochastic policies: PG methods do not need to be programmed in order to combine exploring (select new actions) with exploiting (select known actions). But the stochastic policy lets the agent find the right balance between two
- Disadvantages
  - Local maximum: PG methods converge on a local maximum rather than on the global maximum
  - Hyperparameter sensibility: PG methods suffer from a very high sensibility to hyperparameter tuning and it specially with initialization

4.3 IMPLEMENTATION

Before implementing the previously explained PG's algorithm, some design considerations need to be taken into account:

- Number Neurons per Layer and Number of Layers: The number of inputs and outputs of a neural network will depend directly on the problem. In PG method, the inputs will be the observations of the current State of the Agent in the Environment, and number of output neurons will be the number of Actions that the Agent can perform. However, the number of neurons in the hidden layers is a well-known problem without specific solution. Nonetheless, there exist some common practices:
  - If the problem has a linear solution, there is no need to use hidden layers. However, if it is non-linear problem, more layers will be needed
  - If hidden layers are required, a common practice is to size them in a funnel form, from more neurons in the inputs layers and few in the outputs
  - A common practice used recently, is to use the same size for all the hidden layers

  Using more neurons that what is needed could lead to have an overfitting model. Therefore, it would not be useful due to it could not be generalized to other problems.
  The use of a complex network (deep network) allows to model complex functions using exponentially less neurons than shallow nets, making them much faster to train.

  In the project different network's morphologies will be used in order to find a good policy. Training times and rewards will be evaluated.

- Activation Function:

  Activation Function is the second step that each of the neurons does before sending the output to the next neuron. They first sum or multiply their inputs (input multiplied by the connection weight). Then, the result goes through the activation function which will define the output (Figure 4.6).

  Since the beginning of Neural Networks development, the general thought has been to choose a Sigmoid function (Figure 4.7):

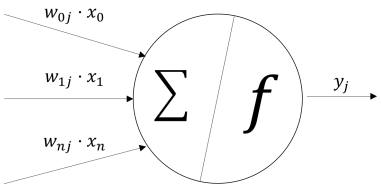

Figure 4.6: Artificial Neuron schema

$$\alpha(z) = \frac{1}{1+e^z} \tag{4.24}$$

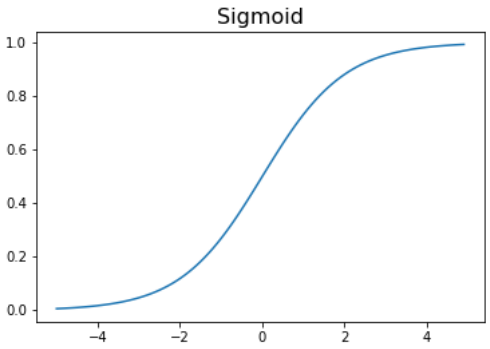

Figure 4.7: Sigmoid activation function

This happened because it was the more similar function to what human brain's neurons use. However, in 2010, Glorot and Bengio's research showed that these could be one of the causes the poor success of Neural Networks. Afterwards, in order not to have saturating activation functions ReLU activation function was used (Figure 4.8):

$$\alpha(z) = max(z,0) \tag{4.25}$$

Despite this improvement, in 2015 a new activation function was proposed, which reduced training time and the neural net performed better. This activation function was the ELU (Figure):

$$ELU_\alpha(z) = \begin{cases} \alpha(exp(z-1) if z < 0 \\ z if z \geq 0 \end{cases} \tag{4.26}$$

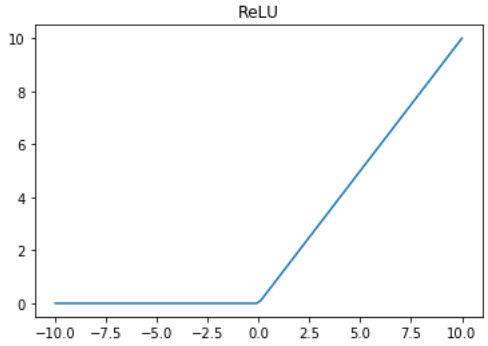

Figure 4.8: ReLU activation function

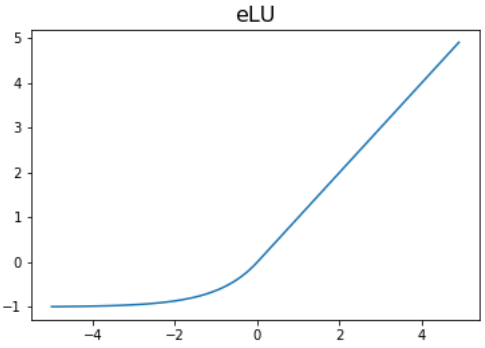

Figure 4.9: ELU activation function

Other activation's functions may be used as Tanh:

$$f(z) = \frac{2}{1+e^{-2x}} - 1 \tag{4.27}$$

However, ELU will be the activation function used in the PG algorithm implementation due to it does not present vanishing problems and it does not have dying units issues as ReLU. It has also added a new hyperparameter to the algorithm (α) that can be tweak in order to improve the model.

- Initialization:

  The initialization of the weights and biases of the Neural Network is also an important step when setting up our algorithm. It could lead to vanishing/exploding gradient problems. Traditionally, the initialization consisted on random initialization using normal distribution with zero mean and standard deviation of one. However, in 2010, Glorot and Bengio that initializing using a normal distribution with mean 0 and standard deviation of:

$$\sigma = \sqrt{2} \times \sqrt{\frac{2}{n_{inputs} + n_{outputs}}} \tag{4.28}$$

This initialization method (Equation 4.28) is called: Xavier initialization.

- Loss Function:

  The Loss Function allows the algorithm to learn. Computing the gradient of this function and using Backpropagation to update the variables, allows the algorithm to improve its performance. There exist multiple loss functions. The election of one respect to another depends on the problem. In our project the Loss Function is defined by our policy method:

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(k)} | s_t^{(k)}) (\hat{Q}_i(s_t^{(k)}, u_t^{(k)}) - V_{\phi i}^{\pi}(s_t^{(k)})) \tag{4.29}$$

  How to obtain the Loss Function is explained in Section 4.2.

  If the Baseline is implemented through the use of the Value Function neural network, the Loss Function will be the Mean Square Error between the reward and the expected reward:

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (R_t - V^{\pi}(s_K^{(i)}))^2 \tag{4.30}$$

4.3.1 Policy Gradient: Implementation and Results

In order to facilitate the implementation and testing of the PG algorithm, it first has been implemented in a OpenAI GYM environment. In this section, a comparison of the results between the implementation of a basic PG algorithm and the PG Baseline will be given. The algorithm implementation was done in the CartPole environment (Figure: 4.10).

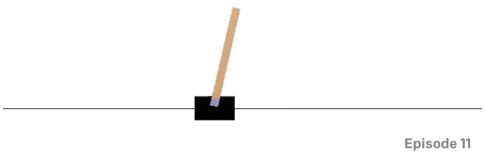

Figure 4.10: OpenAI GYM CartPole

CartPole's description [20]:

A Pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the Cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the Pole remains upright. The episode ends when the Pole is more than 15 degrees from vertical, or the Cart moves more than 2.4 units from the center.

In this scenario, the agent is the Cart, and the Policy is the PG algorithm implemented by us. There are two actions: forces +1 or -1 made to the Cart. ELU Activation function and Xavier Initialization have been used to setup the algorithm.

The Neural Network consisted of:

- 1 input layer of 4 inputs: Cart Position, Cart Velocity, Pole Angle and the Pole Velocity At Tip.
- 1 hidden layer of 16 neurons
- 1 output of 1 neuron (Action 1 or Action 0)

The code implementation of the algorithm in Section 5 has been based on this code. Therefore, the implementation can be check in that Section's explanation.

Results:

Figure 4.11 shows the rewards of the moving average gotten during 2500 episodes implementing the PG basic algorithm. Figure 4.12 shows the rewards of the moving average gotten during 2000 episodes implementing the PG Baseline algorithm.

It can be seen that in the case of the basic PG algorithm it reaches and gets stabilized in reward 200 at episode 1500. However, adding the Baseline in the Loss Function, it gets stabilized in the reward 200 with 500 episodes. It can be seen that implementing the PG algorithm, the Cart is able to learn how to stabilize the Pole by itself. Despite its succeed, it can be seen that reducing the variance, and therefore the number of episodes to reach a solution, by introducing the Baseline, the number of episodes are also reduced.

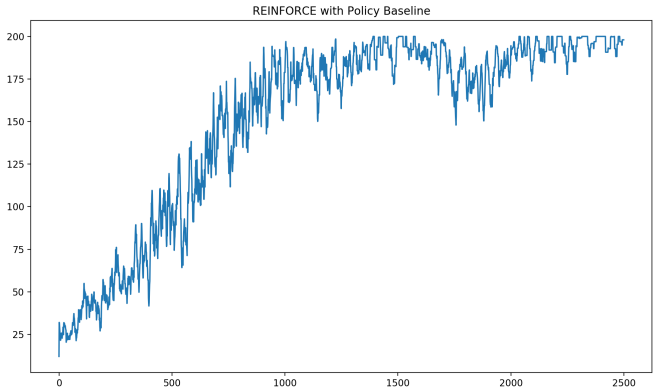

Figure 4.11: Vanilla Policy Gradient reward

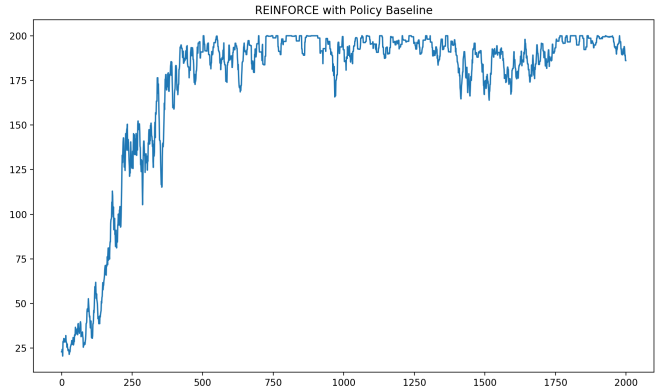

Figure 4.12: Policy Gradient Baseline reward

In conclusion, both algorithms performed as expected. Thus, the implementation of the PG Baseline algorithm has succeeded due to it reduced the number of samples to converge. As it has been said before, this implementation will be used to make our AV move in out simulator (Figure 3.11).

5 DEVELOPMENT

In this section, all the implementation part will be explained. First, an overview of our working environment will be given and then, together with the code, the programming part of the PG algorithm with the simulator setup in Section 3.5. Additionally the code Language and Libraries will be stated.

5.1 ENVIRONMENT

The computer and its specifications used to develop the project were:

- Laptop: Macbook Pro Retina, 13-inch, Early 2015
- Processor: 2,7 GHz Intel Core i5
- RAM Memory: 8 GB
- GPU: Intel Iris Graphics 6100 1536 MB

The programming language used has been Python. It has been used due to the simulator specifications and at the same time because it was the already known language to work with TensorFlow.

- Python version: 3.6.5.

The main Libraries used in order to develop the project has been:

- Numpy: is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays (https://www.numpy.org). Numpy version: 1.16.2
- Matplotlib: Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms (https://matplotlib.org) Matplotlib version: 3.0.3.
- Tensorflow: The core open source library to help the development and training ML models (https://www.tensorflow.org). It is a powerful open source library for numerical computation using data flow graphs. It has been used to to develop the RL algorithm (training and testing). In this project TensorFlow GPU support has not been used. TensorFlow version: 1.8.0.

5.2 Implementation

5.2.1 Introduction

In this section the implementation of the PG algorithm in order to control the host vehicle of the simulator selected will be explained. In order to summary the previous sections, Figure 5.1 shows how the development is done.

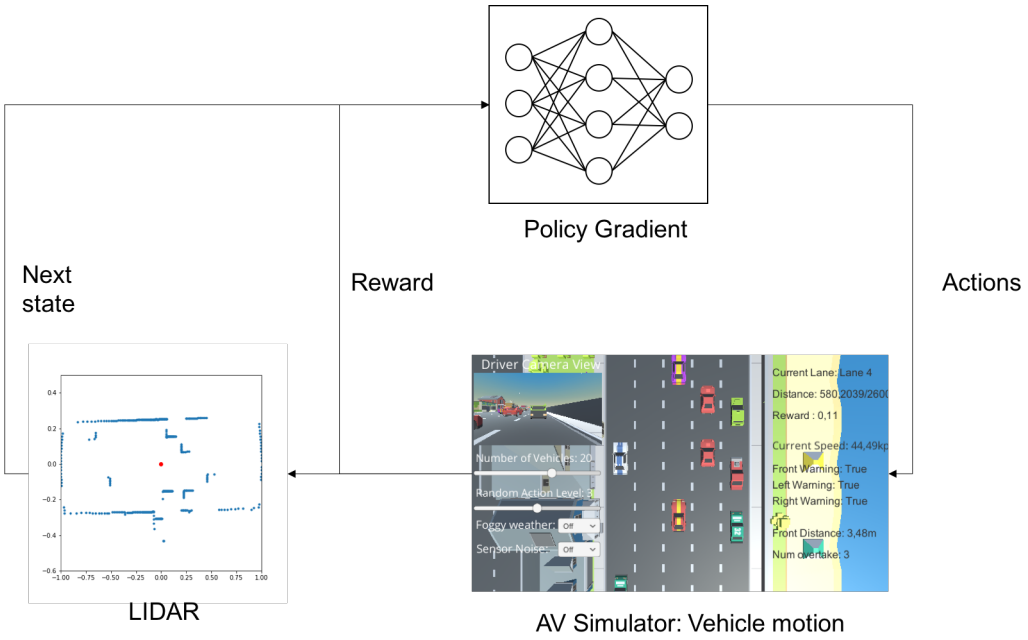

Figure 5.1: Project Layout

The PG algorithm will be the Policy controlling the Agent to move (Actions) in the Environment (AV Simulator). Then, through the LIDAR sensor, the current State will be sent to the agent together with the Reward. Through iterating and updating the Neural Networks' variables by computing the gradients of the Loss Function, the Agent will learn to navigate (combining ADAS functions) on the highway without colliding with other vehicles and maximizing the velocity.

5.2.2 Policy Gradient Configuration

Before the programming's Section, and once the research about ADAS functions, the implementation of the Simulator and PG's methods study is done in the previous sections of the project, these important factors needed to be defined:

- Reward function: It will determine the behaviour of our agent in the highway. Therefore, it is important to set a good reward. However, in order to get better results of the PG algorithm it

is a better to choose a non-complex reward. In this project the main goal for the Agent will be the navigation on the highway as fast as possible (inside the velocity limits), but without having the possibility of colliding with other vehicles.

In the project, it has been set the maximum acceleration the vehicle can achieve to 80km/h and the minimum to 40km/h. Therefore, if the vehicle is going at 80, it will get maximum reward. However, if it is going at 40 it will get 0 reward.

But, in order to assure safety, it will get a penalty whenever it tries to perform a lane change and the Left/Right warnings are activated. Also, if it tries to accelerate when the forward warning is active.

- Number of inputs: The number of inputs will correspond to the number observations of the current state using the sensors. The simulator has two sensors on board. However, in this project only LIDAR's data will be used. One of the main improvements on neural networks' performance is to inject as much prior knowledge as possible into the model in order to speed up the training to converge to a good solution. Thus, in order to make a LC the LIDAR signal will be discretized, and only eight states of the LIDAR's range will be used: (Figure 5.2).

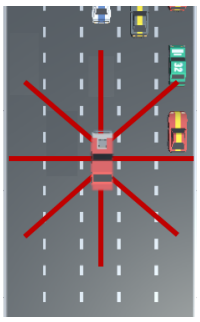

Figure 5.2: LIDAR's observations

Therefore, the number of inputs for both Neural Networks will be 8.

- Number of outputs: The number of outputs of the Policy neural network will correspond to number of actions the vehicle can perform. The Agent (car) can make five actions. Therefore, the number of output neurons will be five. It is important to have in to account, that the policy does not need to know which action (acceleration, deceleration, etc.) corresponds to the vector's position (0,1,...) due to it will learn form the batch episodes rewards.

  The Value function neural network will only have one output corresponding to the expected discounted reward.

Neural Network configuration					
	Input	Hidden layer 1	Hidden layer 2	Hidden layer 3	Output
Conf. 1	8	8	-	-	5
Conf. 2	8	8	8	-	5
Conf. 3	8	8	8	6	5
Conf. 4	8	16	-	-	5

Table 5.1: Neural Network configurations

- Number of layers: As it was said in Section 4.3, there is not a general rule in order to determine the number of hidden layers. Thus, in the project's testing phase, different morphologies has been used in order to see its behaviour (Table 5.1).

- Loss function: There will be two Loss functions; one for the Policy neural network and the other for the Value Function neural network (both Loss functions are explained in Section 4.2):

  – Policy Loss function:

$$\nabla_\theta U(\theta) \simeq \hat{g} = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(k)}|s_t^{(k)})(\hat{Q}_i(s_t^{(k)},u_t^{(k)}) - V_{\phi i}^{\pi}(s_t^{(k)})): \tag{5.1}$$

  – Value function Loss function:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(R_t - V^{\pi}(s_K^{(i)}))^2 \tag{5.2}$$

- Activation function: It will be used a ELU activation functions for both networks (explained in Section 4.3):

```
activation=tf.nn.elu
```

- Initializer: It will be used Xavier initilization:

```
initializer_v = tf.contrib.layers.xavier_initializer()
```

- Batch size: It will determine how many episodes will be run before the update of the weights and biases of the neural networks. In the project a batch size of 10 episodes will be used.

- Number of episodes: An episode is the number of steps the vehicle will be navigating on the highway before returning to initial conditions. However, there are states (collisions) that for

safety reasons will end the episode sooner in order to prevent dangerous situations.

An episode will be of composed by 10000 steps.

- Learning rate and discount rate: Both rates will be changed in order to observe the behaviour of the learning:
  - Learning rate Policy: 0.01
  - Discount rate: 0.95
  - Learning rate Value Function: 0.01

5.3 Programming

All the programming part is based in TF Library. Therefore, the code is divided in a Construction Phase and a Execution Phase:

5.3.1 Construction Phase

First, as in Section 3.5, the Unity ML-Agent environment needs to be initialized, and set the host vehicle's Brain:

```
    env_name = "../environment/jeju_camp.app"
    env = UnityEnvironment(file_name=env_name)
    default_brain = env.brain_names[0]
    brain = env.brains[default_brain]
    env_info = env.reset(train_mode=True)[default_brain]
```

Then, the states of the host vehicle gotten from the LIDAR sensor will be defined using a function:

```
def get_obs_state_lidar(vector_obs):
    state[0] = vector_obs[0,0]      # 0 degrees
    state[1] = vector_obs[0,45]     # 45 degrees
    state[2] = vector_obs[0,90]     # 90 degrees
    state[3] = vector_obs[0,135]    # 135 degrees
    state[4] = vector_obs[0,180]    # 180 degrees
    state[5] = vector_obs[0,225]    # 225 degrees
    state[6] = vector_obs[0,270]    # 270 degrees
    state[7] = vector_obs[0,315]    # 315 degrees
    return state
```

These will define the number of inputs of our Neural Network. Then, in order to define the outputs, the action space needs to be known:

```
    number_actions = brain.vector_action_space_size[0]
    possible_actions = np.arange(0,number_actions,1)
```

The following code will define the Policy Network, and then, using the same morphology, the Value Function Network:

```
with tf.variable_scope("Policy"):
    n_inputs = number_observations
    n_hidden = glob_n_hidden
    n_outputs = number_actions
    # print(n_outputs)
    learning_rate = 0.01
    initializer = tf.contrib.layers.xavier_initializer() # Xavier initialization
```

First, the number of inputs and outputs neurons, the learning rate and initializer are defined. Then, the nodes (placeholders) responsible to output the data that is wanted to feed. Three placeholders will be created: one to feed the neural network input with the states, one with the advantage computed and one with the actions:

```
    state_p = tf.placeholder(tf.float32, shape=[None, n_inputs])  # Tensor with the shape of
     the environment status
    advantage_p = tf.placeholder(tf.float32, shape=[None], name='advantage_p')
    actions_p = tf.placeholder(tf.int32, [None], name='actions_p')
```

Then, the neuron layers are defined:

```
    hidden = tf.layers.dense(state_p, n_hidden, activation=tf.nn.elu, kernel_initializer=
    initializer)
    logits = tf.layers.dense(hidden, n_outputs, activation=None, kernel_initializer=
    initializer)
```

Note that the output node is not a Softmax output. However, it will be done in the next line:

```
    action_probs_p = tf.squeeze(tf.nn.softmax(logits - tf.reduce_max(logits)))
```

Here it is gotten the probability of each of the actions. Then, the loss function is defined:

```
    loss_p = -tf.reduce_mean(tf.log(selected_action_prob) * advantage_p)
```

After this, instead of calling the optimizer to minimize the loss function, first, it is wanted to extract the gradients:

```
    var_w_b_p = tf.trainable_variables(scope_p)
    gradients_p = tf.gradients(loss_p, var_w_b_p)
```

This is because it is wanted to tweak the gradients before optimizing. At each iteration the gradients will be computed running the Policy. However, the weights and biases will be updated computing the mean of the gradients of an episode. In order to do so, the gradients will be sent to the optimizer through the use of placeholders:

```
    optimizer = tf.train.AdamOptimizer(learning_rate)
    training_op = optimizer.apply_gradients(zip(gradient_holder_p, var_w_b_p))
```

Once the Policy neural network has been defined, the Value Function will be defined doing the same as the Policy:

```
    with tf.variable_scope("Value_function"):
    #...
```

However, there will be some differences: There will only be one output (expected value of the discounted reward) and the Loss Function, because it is wanted to minimize the difference between the discounted reward and the expected discounted reward. Thus:

```
    loss_v = tf.reduce_mean(tf.squared_difference(value_estimation, value_real))
```

Mean Square Error function is defined.

Once both Nets have been defined, and before starting the Execution Phase, a function to compute the discounted reward is defined:

```
def discount_rewards(rewards, disc_rate):
    discounted_rewards = np.zeros(len(rewards))
    cumulative_rewards = 0
    for i in reversed(range(0, len(rewards))):
        cumulative_rewards = cumulative_rewards * disc_rate + rewards[i]
        discounted_rewards[i] = cumulative_rewards
    return discounted_rewards
```

5.3.2 Execution Phase

Now that the computational graph has been created in the Construction Phase,the Execution Phase will be explained. It corresponds to the training phase of the algorithm:
First some variables are defined:

```
n_iterations = 200          # number of training episodes / 10
n_max_steps = 10000         # max steps per episode
```

```
n_games_per_update = 10      # batch size
save_iterations = 10         # save the model every 10 training iterations
discount_rate = 0.95         # discount rate reward
```

These variables will be changed to test in order to test and improve the model. Then, the variable initializer is created, as well as the saving node (node responsible in order to be able to save and restore the model):

```
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

In order to evaluate the previously created graph, a session needs to be opened:

```
with tf.Session() as sess:
    init.run()
```

Before starting to run the Policy in the Environment, a Gradient buffer is created to store the gradients across the episodes. Two buffers will be created: one for the Policy network and another for the Value Function network:

```
    grad_buffer_p = sess.run(tf.trainable_variables(scope_p))
    grad_buffer_v = sess.run(tf.trainable_variables(scope_v))
    for i, g in enumerate(grad_buffer_p):
        grad_buffer_p[i] = g * 0
    for i, g in enumerate(grad_buffer_v):
        grad_buffer_v[i] = g * 0
```

Now the training part starts. There will be a general iteration that defines how many episodes we want out training lasts (divided by 10). Then, inside this iteration there will be another iteration that will be the responsible to update the weights and biases of the networks, which will be the batch size. And finally, there will be the iteration that defines how many steps each episode lasts.

```
for iteration in range(n_iterations):
    for game in range(n_games_per_update):
        for step in range(n_max):
```

Before starting the inner iteration (one episode), the environment needs to be reset and get the first state from the LIDAR, so at every episode the host vehicle starts again:

```
env_info = env.reset(train_mode=train_mode)[default_brain]
obs_0 = get_obs_state_lidar(env_info.vector_observations)
```

The first step in this iteration is to run the Policy and to choose a random action based on the probability given:

```
action_probs = sess.run(action_probs_p, feed_dict={state_p: obs_0.reshape(1, n_inputs)})
action = np.random.choice(possible_actions, p=action_probs)
```

Once we have the action and the actual state, the Value Function is run in order to get the estimate discounted reward:

```
est_value = sess.run(value_estimation, feed_dict={state_v: obs_0.reshape(1, n_inputs)})
```

Then, with the previous action obtained, we send it to the environment, getting as a return the actual state of the vehicle and the reward:

```
env_info = env.step(action)[default_brain]
```

In order to save relevant information at each step (the previous and next LIDAR observation, the action taken, the estimated value of the reward, the actual reward and the velocity at that state) a buffer was created:

```
episode_log.append([obs_0, action, est_value, r, obs_1, vel])
```

At the end of the episode, the discounted rewards and the difference between this reward and the estimated by the value function are computed:

```
current_rewards_discount = discount_rewards(episode_log[:, 3], discount_rate)
advantage = current_rewards_discount - episode_log[:, 2]
```

And the gradients of both networks:

```
gradient_p = sess.run(gradients_p, feed_dict={
    state_p: np.vstack(episode_log[:, 0]),
    actions_p: episode_log[:, 1],
    advantage_p: advantage})
 gradients_val = sess.run(gradients_v, feed_dict={
    state_v: np.vstack(episode_log[:, 0]),
    value_real: current_rewards_discount})
```

Finally, the update of weights and biases is done after *n_games_per_update* episodes:

```
sess.run(training_op, feed_dict=(dict(zip(gradient_holder_p, grad_buffer_p))))
sess.run(training_op_v, feed_dict=(dict(zip(gradient_holder_v, grad_buffer_v))))
```

Additionally, a counter has been added in order to save the model every 10 iterations. It is a checkpoint in order to have a security copy of the parameters in case a model restoring is required:

```
if iteration % save_iterations == 0:
    saver.save(sess, "./car_{}.ckpt".format(iteration))
```

Here concludes the integration of the PG algorithm with the highway Simulator. In order to be able to follow the process during the execution, some additional plotting functions have been implemented. These functions were plotting the reward function, the percentage of each of the actions between the others per episode and the mean velocity of the vehicle in each episode. They will be shown in the Results Section (6).

In the following section, the results of this implementation will be shown.

6 RESULTS

In this section the final results gotten by running the PG algorithm implemented by us, and interacting with the highway simulator will be shown.

Different parameters and network morphologies have been used in order to see how the vehicles were behaving, and how long it would take to learn a proper policy. Table 5.1 shows the different network's shapes used. Moreover, other parameters as learning rate, number of steps per episode have been changed. Table 6.1 shows the different training's configurations performed with its training's times.

TRAINING						
	Hidden layers	Hidden neurons	Learning rate	Number steps episode	Reward	Running time
TRAIN 1	1	8	0.01	200-1000	Velocity ↓ LC	1h51min
TRAIN 2	1	8	0.01	1000	Velocity ↓ LC	1h23min
TRAIN 3	1	8	0.01	10000	Velocity ↓ LC	2h45min
TRAIN 4	1	8	0.01	10000	Velocity	9h03min
TRAIN 5	1	8	0.01	10000	Velocity	27h01min
TRAIN 6	1	8	0.01	10000	Velocity	27h34min
TRAIN 7	1	8	0.01	10000	Velocity	12h14min
TRAIN 8	2	8-6	0.01	10000	Velocity	29h48min
TRAIN 9	2	8-6	0.05	10000	Velocity	20h51min
TRAIN 10	1	8-6	0.001	10000	Velocity	22h25min
TRAIN 11	1	16	0.01	10000	Velocity	18h36min
TRAIN 12	1	16	0.01	10000	Velocity	29h27min

Table 6.1: Training Configurations

In order to evaluate each of the configurations, the following parameters have been saved during the training, and plotted in Figures of three graphs:

- Velocity: shows the mean velocity at each episode in km/h. This velocity should be increased during the training as a consequence of the policy
- Reward: shows the reward of the moving average. The reward gotten from the environment should show a increasing tendency as a consequence of the increase of velocity

- Action taken: it shows the percentage of the actions taken between the five actions in an episode. Where: acceleration is the green line, deceleration red, do nothing the doted black line, LC left is the purple and the LC right is the blue.

Before taking a look at the results' graphs, it should be taken into account that velocity may sharply vary in some episodes due to the traffic conditions (e.g. the vehicle gets stuck between other vehicles and it can not perform any movement to go faster). Figure 6.1 shows a traffic situation where the mean velocity of that episode may not be the expected due to this jam restrictions.

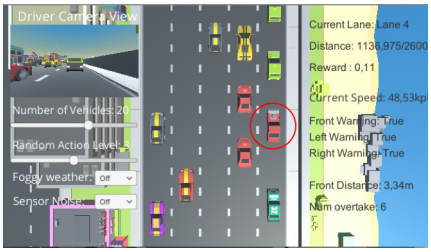

Figure 6.1: Traffic jam

TRAIN 8 configuration has been the model that has achieved a higher mean velocity (being only the velocity the reward). It consists of two hidden layers with 8 and 6 neuron (form input to output). Observing Figure 6.2, it can be seen that it reaches 70km/h after 200 episodes. However, until episode 300, the percentage of chosen actions will not be steady. It can also be seen, that the acceleration action (green line) at the beginning (when parameters of both neural networks are initialized) is not the action taken more times. However, during the training it increases until to reach a stationary state being the 75% of the times the action chosen. Additionally, the deceleration action is the action being reduced more rapidly as well as both lane change action. Do nothing action is taken in a percentage of 25%.In order to get these results, the training's time has been of 29 hours and 28 minutes. Therefore, it can be seen that the PG algorithm is working, due to it is increasing the selection of the acceleration action and decreasing the deceleration action.

Other training configurations with significant results are:

Figure 6.3 shows the results of configuration TRAIN 9, which is the same configuration than TRAIN 8 except that the learning rate has been increased to 0.05. It can be seen that the learning of the actions is much more unstable. It oscillates between actions and at the end, it reaches a bad solution

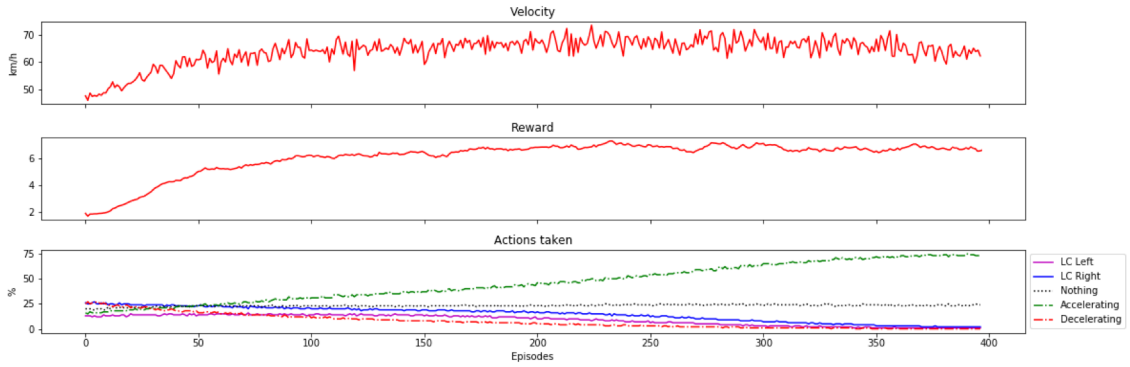

Figure 6.2: Train 8

of only performing left LC. It can be seen that the reward, as well as the velocity, is not increasing at initial states, and when it starts to mainly perform left LC actions, it suddenly starts to decrease.

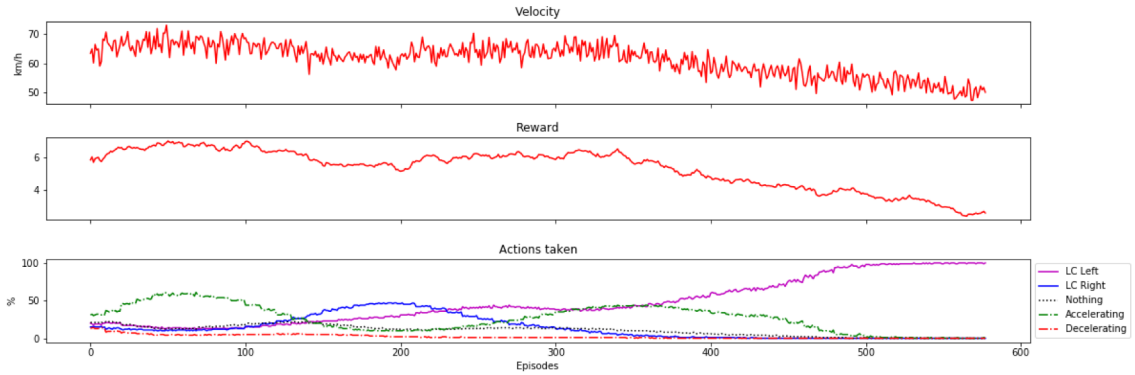

Figure 6.3: Train 9

Another significant configuration, and opposite to the previous one, is TRAIN 10. In this case, the learning rate has been set at 0.001. In Figure 6.5 can be seen how the reward remains steady as well as the actions and the velocity. It is because its learning time is much longer than in the other cases. However, it can be seen how the acceleration is slightly increasing.

First training configurations implemented was TRAIN 1. The difference with the others was the reward function. It was not only trying to maximize the velocity, but was trying to reduce the number of LC of the vehicle in order to decrease this movement making it more similar to human behaviour. However, and because it was a complex reward, the agent was not able to learn what exactly the reward function was giving. Figure 6.5 shows how the policy learns that the best action to perform is: do nothing. The velocity plot shows that the velocity was increasing at the beginning,

Figure 6.4: Train 10

when the acceleration action was the more taken. However, after 300 episodes, when all actions except do nothing action decreased to 0, the velocity and the reward decreased.

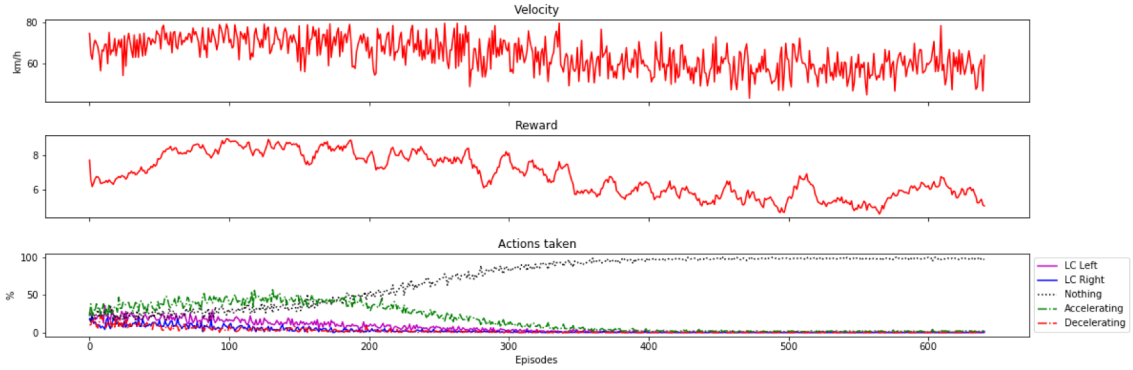

Figure 6.5: Train 1

TRAIN 5 configuration (Figure 6.6), was a neural network with only one hidden layer. The reward was only the velocity, without any collision penalty. However, with this configuration the agent learned that the best policy would be to only accelerate. It can be seen that the rest of actions tend to zero. Also, the reward function was increasing at the beginning, however, after not performing any other movement rather than accelerate, it started to decrease.

TRAIN 11 was done by increasing the number of neurons, of the only hidden layer, from 8 to 16. The results gotten (Figure 6.7) were showing that the tendency was to only accelerate the car. After 800 iterations all actions except acceleration were decreased to zero.

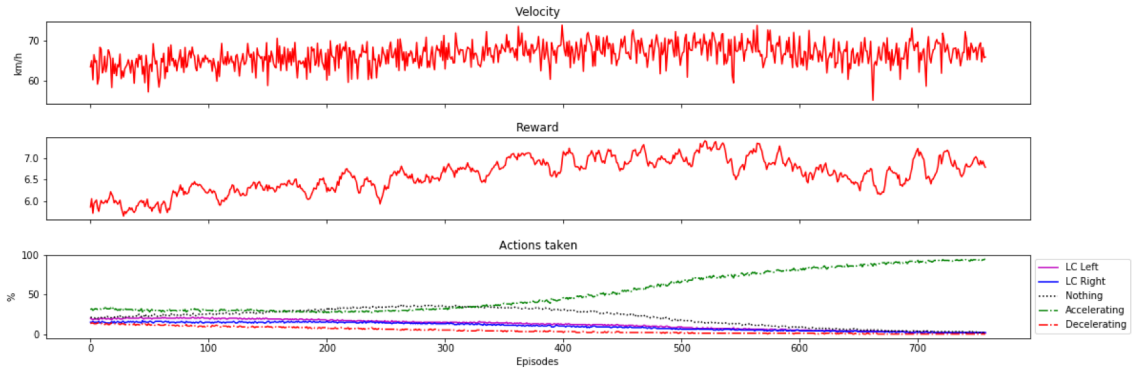

Figure 6.6: Train 5

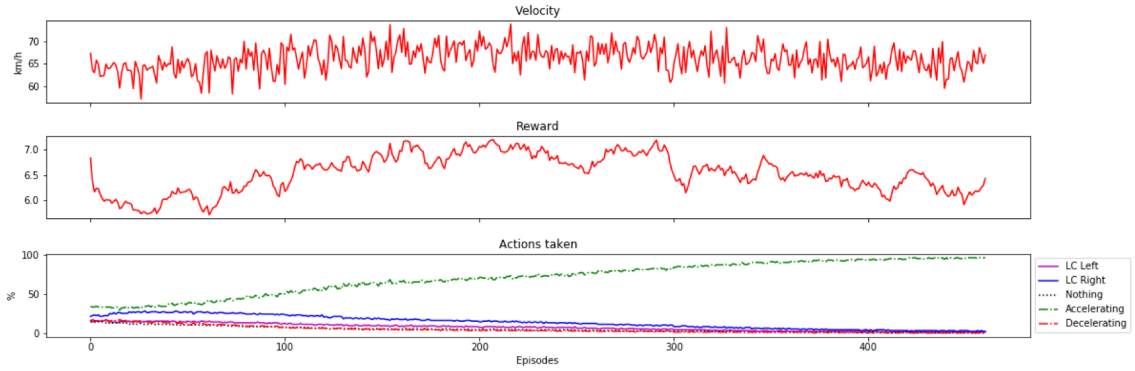

Figure 6.7: Train 11

7 CONCLUSIONS

The project had the main objective of making an Autonomous Vehicle navigate on a highway by combining ADAS functions using Reinforcement Learning. After the execution of this project, it can be seen that the objective has been achieved. It has been possible by first having succeeded with the specific objectives set at the beginning.

In order to have a better understanding on how to develop vehicle's control systems, the Regulatory Framework has been explained. It made a review about which Regulations were regulating the automated functions. It has been seen that UN-Regulation 79 (Steering equipment) is, nowadays, the regulation where all ADAS's requirements related to steering are stated. Moreover, in order to be informed about the future of AV in terms of regulation, it will be mandatory to be updated about the work in UNECE's Working Group GRVA.

The first step for the implementation of the RL algorithm was to find a simulator that fit with our project. It could be seen that there exist several simulators depending on their purpose; from professional use simulators, to educational. After a comparison between ten of these simulators, Unity ML-Agents Highway Simulator was found to be the best for the project. It is a five lane highway simulator with more vehicles on the road. Additionally, and in order to facilitate future implementations, a guideline and a simulator interaction example was given.

Finally, it was wanted to implement a RL algorithm to control the vehicle. Policy gradient was RL method chosen to develop the project using TensorFlow Library. The implementation consisted not only on a basic Policy Gradient algorithm, but also on the implementation of a Baseline in order to reduce the number of samples needed have a solution. Both were implemented and testes successfully using the CartPole environment.

Once all the sub-objectives were achieved, the implementation of the simulator with the RL algorithm was done. The simulator worked as expected. The Policy Gradient algorithm implemented was allowed the vehicle to navigate on a highway at a mean speed of 70km/h and not colliding with other vehicles.

The project is useful not only in order to develop and test a Reinforcement Learning algorithm in an Autonomous Vehicle environment, but also to understand how Regulatory framework determines the course for the Manufacturers in ADAS and AV development and launch to the market.

7.1 Future Work

The project can be used as basis for a future projects. The development of the project from scratch has allowed documenting all steps taken during the implementation of both, the simulator and the PG algorithm. Additionally, a guideline of simulator's installation and interaction has been provided (with code and examples), which will allow a faster implementation and understanding of it.

However, and due to the wide scope of the project, many improvements and test has been left for the future. On one hand, a deeper analysis of a better neural network morphology can be done in order to enhance results in training time and velocity. Also, tweaking more the different training parameters (learning rate, discount rate, etc.) can also lead to better results.
On the other hand, others RL algorithms such as Q-Learning could be implemented in order to compare results.

Furthermore, and using more Simulator's resources, the camera data could be used in order to give more information about the current state of the vehicle. Also, a Multi-Agent could be implemented in order to train faster the Policy, due to the simulator allows to control not only the host vehicle, but other vehicles.

Finally, by being update in UNECE Working Group GRVA, new ADAS functions may be implemented in order to improve the AV navigation.

8 Cost

8.1 Time

The time's cost of the project is resumed in the following Gantt Diagram 8.1. The Grantt Diagram is divided in the three main topics discussed in the project (divided in their specific sub tasks) and the Results. The total development time of the project has been ten months. However, only the last two months and half of February were worked full time, due to the first phases of the project were done together with an internship. Thus, the remainder months (from July to January) were worked as partial time.

It can be seen that the project has taken most of the time in the simulator's research. It was because it presented some difficulties on implementing and testing whether or not their were proper simulators for the project.

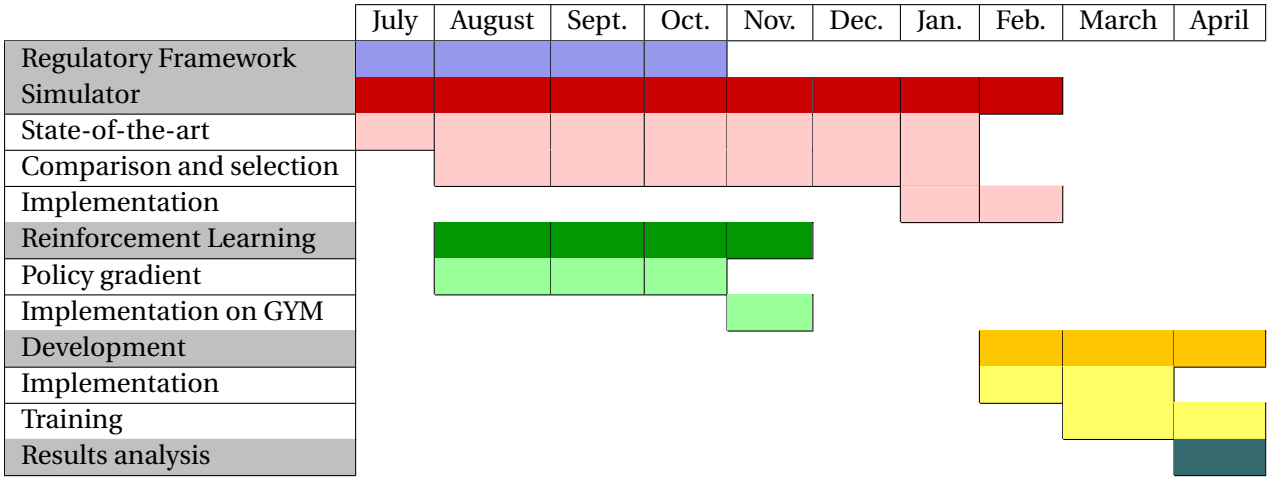

Table 8.1: Gantt Diagram

8.2 Budget

This Section wants to give a brief economic analysis of the development of this project by a Master student. Taking into account the number of hours dedicated to the project and the material used, the total cost of its implementation has been of: 12494€. Table 8.2 and Table 8.3 breaks-down the budget between the material costs (Hardware and Software) and the personal cost.

Concept	Units	Cost (€)
Equipment		
Macbook Pro Retina 2015	1	1300
Software		
Simulator	1	0
TOTAL		1300

Table 8.2: Material Cost

The material used to implement the project has been a Macbook Pro Retina, 13-inch, Early 2015. The project can be implemented using other laptops which fills simulator's requirements. The simulator used was an open-source software which had no cost.

The personal cost is computed taking into account the number of hours worked on the project (Table 8.1) and a salary equal to the cost per hour paid by the companies for a Master student internship: 8,36€/h (based on personal experience). Additionally, as the Regulatory Section explained is based on the experience I had as Homologation Engineer, the cost is approximate to the cost charged by the company for a presentation about this topic: 2500€.

Concept	Hours	Hour price (€/h)	Cost (€)
Regulatory research	120	8.36	1003
Regulations AV presentation	-	-	2500
Simulator	400	8.36	3344
Reinforcement Learning	120	8.36	1003
Development	400	8.36	3344
TOTAL			11195

Table 8.3: Personal Cost

9 ENVIRONMENTAL IMPACT

The implementation of Autonomous Vehicles may change how mobility is understood nowadays. It will switch from particular vehicles to shared vehicles. People will not longer be the owner of the vehicle, but each time they need a ride, they will hire its service. Vehicles will be more connected and coordinated among them. For these reasons, Autonomous Vehicles will not only have an environmental impact, but also a socioeconomic impact.

It is reasonable to say that due to people will not longer own a vehicle, it will decrease the number of vehicles. Thus, it will have a positive environmental impact due to the reduction of energy consumption by the vehicles. However, despite the energy used by AV will be mainly electric with the use of batteries, the contamination of these will be conditioned to the use of renewable energies.

Also, the fact that vehicles will be more connected and coordinated between them, it will lead to a more fluent traffic road where traffic accidents will be reduced. Additionally, routs will be optimized and predicted in advance. Therefore, traffic jams will be decreased as well.

However, there exists some socioeconomic impacts as well that may may affect negatively society. The fact that vehicles will not be controlled by humans, it will have the advantage of decreasing the number of accidents, but at the same time it will take a lot of jobs related with transportation.

Finally, society will face another change in its daily behaviour. They will be using the vehicles not as a driver or as a passenger, but as a user, leading to have the opportunity to spend that time doing more quotidian tasks.

In summary, this project might be a little step for the development of Autonomous Vehicles, and thus, to have a good environmental and socioeconomic impact on our society.

REFERENCES

[1] Organización Mundial de la Salud *Las 10 principales causas de defunción*. https://www.who.int/es/news-room/fact-sheets/detail/the-top-10-causes-of-death

[2] United States Environmental Protection Agency *Total US gas emissions by sector in 2016*. https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions

[3] INTRIX *Global Traffic Scorecard*. http://inrix.com/scorecard/

[4] National Traffic Highway Safety Administration (NHTSA) *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115

[5] Society of Automotive Engineers *Levels of Driving Automation - Standard for Self-Driving Vehicles*. https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-“levels-of-driving-automation”-standard-for-self-driving-vehicles

[6] UNECE *WP29: World Forum for Harmonization of Vehicle Regulations*. https://www.unece.org/trans/main/wp29/introduction.html

[7] EURLEX *Directive 2007/46/EC of the European Parliament and of the Council*. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32007L0046

[8] European Comission *Type approval of vehicles*. http://ec.europa.eu/growth/sectors/automotive/technical-harmonisation/faq-auto_en

[9] UNECE World Forum for Harmonization of Vehicle Regulations *Exchange of views on vehicle automation related work priorities*. www.unece.org%2Ffileadmin%2FDAM%2Ftrans%2Fdoc%2F2018%2Fwp29grva%2FECE-TRANS-WP.29-GRVA-2018-01e.docx&usg=AOvVaw3jVYhdBlEjbQt9g3zspfVY

[10] EUROPEAN COMMISSION *Annex 1: Strategic Action Plan on Road Safety*. https://eur-lex.europa.eu/resource.html?uri=cellar%3A0e8b694e-59b5-11e8-ab41-01aa75ed71a1.0003.02/DOC_2&format=PDF

[11] DeepMind *AlphaGo*. https://deepmind.com/research/alphago/

[12] Massachusetts Institute of Technology *MIT 6.S091: Introduction to Deep Reinforcement Learning*. https://deeplearning.mit.edu

[13] Xavier Glorot and Yoshua Bengio *Understanding the difficulty of training deep feedforward neural networks*. DIRO, Université de Montréal, Montréal, Québec, Canada

[14] Bing Xu, Naiyan Wang, Tianqi Chen and Mu Li *Empirical Evaluation of Rectified Activations in Convolution Network*. arXiv:1505.00853v2 [cs.LG] 27 Nov 2015

[15] Speaker: Pieter Abbeel. Authors: John Schulman and Xi (Peter) Chen *Deep RL Bootcamp Lecture 4A: Policy Gradients*. https://www.youtube.com/watch?v=S_gwYj1Q-44

[16] Andrej Karpathy *Lecture 4B Deep RL Bootcamp: Policy Gradients Revisited: Pong from Pixels*. https://www.youtube.com/watch?v=tqrcjHuNdmQ

[17] Ronald J. Williams *Simple Statistical Gradient-Following Algorithm for Connectionist Reinforcement Learning*. College of Computer Science, Northeastern University, Boston, MA 02115

[18] Kyushik Min and Hayoung Kim *DRL Based Self Driving Car Control*. https://github.com/MLJejuCamp2017/DRL_based_SelfDrivingCarControl

[19] Dr. Danny Lange *Unity ML-agent*. https://unity3d.com/machine-learning

[20] OpenAI *GYM CartPole-v1*. https://gym.openai.com/envs/CartPole-v1/